ForValueNet

Andrew Robinson
Department of Mathematics and Statistics
University of Melbourne
Parkville, Vic. 3010
A.Robinson@ms.unimelb.edu.au

June 10–12 2010

Contents

Chapter 1

Data

Strategies for convenient data manipulation are the heart of the R experience. The object orientation ensures that useful and important steps can be taken with small, elegant pieces of code.

We spend time on this very basic data-handling material because the time that we spend now will greatly ease the handling of your own data.

1.1 Data: Objects and Classes

So, what does it mean to say that R is object oriented? Simply, it means that all interaction with R is through objects. Data structures are objects, as are functions, as are scripts. This seems obscure right now, but as you become more familiar with it you'll realize that this allows great flexibility and intuitiveness in communication with R, and also is occasionally a royal pain in the bum.

For the most part, object orientation will not change the way that we do things, except that it will sometimes make them easier than we expect. However, an important consequence of object orientation is that all objects are members of one or more classes.

We create objects and assign names to them using the left arrow: "<-". R will guess what class we want the object to be, which affects what we can do with it. We can change the class if we disagree with R's choice.

```
> a <- 1                    # Create an object "a" and
>                           #    assign to it the value 1.
> a <- 1.5                  # Wipe out the 1 and make it 1.5 instead.
> class(a)                  # What class is it?

[1] "numeric"

> class(a) <- "character" # Make it a character
> class(a)                  # What class is it now?

[1] "character"

> a

[1] "1.5"
```

```
> a <- "Andrew"           # Wipe out the 1.5 and make it "Andrew" instead.
> b <- a                  # Create an object "b" and assign to it
>                         #    whatever is in object a.
> a <- c(1,2,3)           # Wipe out the "Andrew" and make it a vector
>                         #    with the values 1, 2, and 3.
>                         #    Never make c an object!
> b <- c(1:3)             # Wipe out the "Andrew" and make it a vector
>                         #    with the values 1, 2, and 3.
> b <- mean(a)            # Assign the mean of the object a to the object b.
> ls()                    # List all user-created objects

[1] "a" "b"

> rm(b)                   # Remove b
```

A couple of points are noteworthy: we didn't have to declare the variables as being any particular class. R coerced them into whatever was appropriate. Also we didn't have to declare the length of the vectors. That is convenient for the user.

Exercise 1 Instantiation

Using the code above, or similar, familiarize yourself with the process of creating and destroying objects, printing them, and checking their class.

1.2 Classes of Data

There are two fundamental kinds of data: numbers and strings (anything that is not a number is a string). There are several types of strings, each of which has unique properties. R distinguishes between these different types of object by their *class*.

R knows what these different classes are and what each is capable of. You can find out what the nature of any object is using the `class()` command. Alternatively, you can ask if it is a specific class using the `is.`*`className`*`()` command. You can often change the class too, using the `as.`*`className`*`()` command. This process can happen by default, and in that case is called *coercion*.

1.2.1 Numeric

A number. Could be a integer or a real number. R can generally tell the difference between them using context. We check by `is.numeric()` and change to by `as.numeric()`. R also handles complex numbers, but they're not important for this course. We can do all the usual things with numbers:

```
> a <- 2                # create variable a, assign the number 2 to it.
> class(a)              # what is it?

[1] "numeric"

> is.numeric(a)         # is it a number?

[1] TRUE
```

```
> b <- 4                    # create variable b, assign the number 4 to it.
> a + b                     # addition

[1] 6

> a - b                     # subtraction

[1] -2

> a * b                     # multiplication

[1] 8

> a / b                     # division

[1] 0.5

> a ^ b                     # exponentiation

[1] 16

> (a + b) ^ 3               # parentheses

[1] 216

> a == b                    # test of equality (returns a logical)

[1] FALSE

> a < b                     # comparison (returns a logical)

[1] TRUE

> max(a,b)                  # largest

[1] 4

> min(a,b)                  # smallest

[1] 2
```

1.2.2 String

A collection of one or more alphanumerics, denoted by double quotes. We check whether or not our object is a string by `is.character()` and change to by `as.character()`. R provides numerous string manipulation functions, including search capabilities.

```
> a <- "string"         # create variable a, assign the value "string" to it.
> class(a)              # what is it?

[1] "character"

> is.numeric(a)         # is it a number?

[1] FALSE

> is.character(a)       # is it a string?

[1] TRUE

> b <- "spaghetti"      # create variable b, assign the value "spaghetti" to it.
> paste(a, b)           # join the strings

[1] "string spaghetti"

> paste(a, b, sep="")   # join the strings with no gap

[1] "stringspaghetti"

> d <- paste(a, b, sep="")
> substr(d, 1, 4)       # subset the string

[1] "stri"
```

1.2.3 Factor

Factors represent categorical variables.

In practice, factors are not terribly different than strings, except they can take only a limited number of values, which R keeps a record of, and R knows how to do very useful things with them. We check whether or not an object is a factor by `is.factor()` and change it to a factor, if possible, by `factor()`.

Even though R reports the results of operations upon factors by the levels that we assign to them, R represents factors internally as an integer. Therefore, factors can create considerable heartburn unless they're closely watched. This means: whenever you do an operation involving a factor you must make sure that it did what you wanted, by examining the output and intermediate steps.

```
> a <- c("A","B","A","B")   # create vector a
> class(a)                  # what is it?

[1] "character"

> is.character(a)           # is it a string?
```

```
[1] TRUE
> is.factor(a)              # is it a factor?
[1] FALSE
> a <- factor(a)            # make it so
> levels(a)                 # what are the levels?
[1] "A" "B"
> table(a)                  # what are the counts?
a
A B
2 2
> a <- factor(c("A","B","A","B"), levels=c("B","A"))
```

Sometimes it will be necessary to work with a subset of the data, perhaps for convenience, or perhaps because some levels of a factor represent irrelevant data. If we subset the data then it will often be necessary to redefine any factors in the dataset, to let R know that it should drop the levels that are missing.

There will be much more on factors when we start manipulating vectors (Section 1.3.1).

1.2.4 Logical

A special kind of factor, that has only two levels: True and False. Logical variables are set apart from factors in that these levels are interchangeable with the numbers 1 and 0 (respectively) via coercion. The output of several useful functions are logical (also called boolean) variables. We can construct logical statements using the and (&), or (|), not (!) operators.

```
> a <- 2                # create variable a, assign the number 2 to it.
> b <- 4                # create variable b, assign the number 4 to it.
> d <- a < b            # comparison
> class(d)              # what is it?
[1] "logical"
> e <- TRUE             # create variable e, assign the value TRUE to it.
> d + e                 # what should this do?
[1] 2
> d & e                 # d AND e is True
[1] TRUE
> d | e                 # d OR e is also True
[1] TRUE
> d & !e                # d AND (NOT e) is not True
[1] FALSE
```

We can ask for the vector subscripts of all objects for which a condition is true via `which()`.

1.2.5 Missing Data

The last and oddest kind of data is called a missing value (NA). This is not a unique class, strictly speaking. They can be mixed in with all other kinds of data. It's easiest to think of them as place holders for data that should have been there, but for some reason, aren't. Unfortunately their treatment is not uniform in all the functions. Sometimes you have to tell the function to ignore them, and sometimes you don't. And, there are different ways of telling the function how to ignore them depending on who wrote the function and what its purpose is.

There are a few functions for the manipulation of missing values. We can detect missing values by `is.na()`.

```
> a <- NA                # assign NA to variable A
> is.na(a)               # is it missing?

[1] TRUE

> class(a)               # what is it?

[1] "logical"

> a <- c(11,NA,13)       # now try a vector
> mean(a)                # agh!

[1] NA

> mean(a, na.rm=TRUE)    # Phew! We've removed the missing value

[1] 12

> is.na(a)               # is it missing?

[1] FALSE  TRUE FALSE
```

We can identify the complete rows (*i.e.* rows that have no missing values) from a two-dimensional object via the `complete.cases()` command.

Exercise 2 Run the Preceding Code

Run each of the preceding swatches of code, and examine the output. Make sure that you understand what each individual piece is doing.

1.3 Structures for Data

Having looked at the most important data types, let's look at the mechanisms that we have for their collective storage and manipulation. There are more than we cover here - some of which (matrix, list) can be very useful.

1.3.1 Vector

A vector is a one-dimensional collection of atomic objects (atomic objects are objects which can't be broken down any further). Vectors can contain numbers, characters, factors, or logicals. All the objects that we created earlier were vectors, although some were of length 1. The key to vector construction is that all the objects must be of the same class. The key to vector manipulation is in using its subscripts. The subscripts are accessed by using the square brackets [ ].

```
> a <- c(11,12,13) # a is a vector
> a[1]             # the first object in a

[1] 11

> a[2]             # the second object in a

[1] 12

> a[-2]            # a, but without the second object

[1] 11 13

> a[c(2,3,1)]      # a, but in a different order

[1] 12 13 11

> a + 1            # Add 1 to all the elements of a

[1] 12 13 14

> length(a)        # the number of units in the vector a

[1] 3

> order(c(a,b))    # return the indices of a and b in increasing order

[1] 4 1 2 3

> c(a,b)[order(c(a,b))] # return a and b in increasing order

[1]  4 11 12 13

> a <- c(11,NA,13) # a is still a vector
> a[!is.na(a)]     # what are the elements of a that aren't missing?

[1] 11 13

> which(!is.na(a)) # what are the locations of the non-missing elements of a?

[1] 1 3
```

Notice that in `a[!is.na(a)]`, for example, we were able to nest a vector inside the subscript mechanism of another vector! This example also introduces a key facility in R for efficient processing: vectorization.

Vectorization

The concept underlying vectorization is simple: to make processing more efficient. Recall that in section 1.2.5, when we applied the `is.na()` function to the vector `a` it resulted in the function being applied to *each element* of the vector, and the output itself being a vector, without the user needing to intervene. This is vectorization.

Imagine that we have a set of 1,000,000 tree diameters and we need to convert them all to basal area. In C or Fortran we would write a loop. The R version of the loop would look like this (wrapped in a timer).

```
> diameters <- rgamma(n=1000000, shape=2, scale=20)
> basal.areas <- rep(NA, length(diameters))
> system.time(
+             for (i in 1:length(diameters)) {
+               basal.areas[i] <- diameters[i]^2 * pi / 40000
+             }
+ )

   user  system elapsed
  5.483   0.028   5.540
```

That took just over three seconds on my quite old computer. However, if we vectorize the operation, it becomes considerably faster.

```
> system.time(
+             basal.areas <- diameters^2 * pi / 40000
+             )

   user  system elapsed
  0.066   0.019   0.086
```

It's about forty to fifty times faster. Of course, had we programmed this function in C or Fortran, the outcome would have been much faster still. The R programming mantra might be: compile only if you need to, loop only if you have to, and vectorize all the time.

Vectorization only works for some functions; *e.g.* it won't work for `mean()`, because that would make no sense; we want the mean of the numbers in the vector, not the mean of each individual unit. But, when vectorization works it makes life easier, code cleaner, and processing time faster.

Note that `pi` is a single value, and not of length 10^6, and R assumed that we would like it repeated 10^6 times. This is called recycling.

Recycling

You may have noticed above that R provided a convenience for the manipulation of vectors. When we typed

```
> a <- c(11, 12, 13)
> a + 1

[1] 12 13 14
```

R assumed that we wanted to add 1 to each element of `a`. This is called recycling, and is usually very useful and occasionally very dangerous. R recycled `1` until it had the same length as `a`. interpreted the function as:

```
> a + c(1, 1, 1)

[1] 12 13 14
```

For a further example, if we want to convert all the numbers in a vector from inches to centimetres, we simply do

```
> (a <- a * 2.54)

[1] 27.94 30.48 33.02
```

Recycling can be dangerous because sometimes we want the dimensions to match up exactly, and mismatches will lead to incorrect computations. If we fail to line up our results exactly - *e.g.* because of some missing values - R will go ahead and compute the result anyway. The only way to be sure is to watch for warnings, examine the output carefully, and keep in mind that most of the time this facility is really helpful.

```
> a + c(1, -1)

[1] 28.94 29.48 34.02
```

Exercise 3 Vectors can be numbers too ...

Create a vector of tree diameters (in cm), say,

```
> diameters <- c(26.4, 43.7, 56.0, 33.5)
```

1. Convert them all to basal area (g, in m^2) in one command[1].
2. Find the largest basal area.
3. Find the position in the vector of the largest basal area.

1.3.2 Dataframe

A dataframe is a powerful two-dimensional vector-holding structure. It is optimized for representing multidimensional datasets: each column corresponds to a variable and each row corresponds to an observation. A dataframe can hold vectors of any of the basic classes of objects at any given time. So, one column could be characters whilst another could be a factor, and a third be numeric.

We can still refer to the objects within the dataframe through their subscripts: using the square brackets. Now there are two dimensions: row, and column. If either is left blank, then the whole dimension is assumed. That is, `test[1:10,]` will grab the first ten rows of all the columns of dataframe test, using the above-noted expansion that the colon fills in the integers. `test[,c(2,5,4)]` will grab all the rows for only the second,

[1] $g = \frac{\pi d^2}{40000}$

fifth and fourth columns. These index selections can be nested or applied sequentially. Negative numbers in the index selections denote rows that will be omitted.

Each column, or variable, in a dataframe has a unique name. We can extract that variable by means of the dataframe name, the column name, and a dollar sign as: *dataframe$variable*.

A collection of columns can be selected by name by providing a vector of the column names in the index specification. This is useful in dataframes in which the locations of the columns is uncertain or dynamic, but the names are fixed.

We can also use the `subset` function to excerpt the pieces that we need.

If a comma-delimited file is imported, then R will assume that it is meant to be a dataframe. The command to check is `is.data.frame()`, and the command to change it is `as.data.frame()`. There are many functions to examine dataframes; we showcase some of them below.

```
> ufc <- read.csv("../data/ufc.csv")      # ufc is a dataframe
> is.data.frame(ufc)                      # we hope

[1] TRUE

> dim(ufc)                                # the size of the dimensions (r,c)

[1] 637   5

> names(ufc)                              # the labels of the columns

[1] "plot"    "tree"    "species" "dbh"     "height"

> ufc$height[1:5]                         # first 10 heights

[1]  NA 205 330  NA 300

> ufc$species[1:5]                        # first 10 species

[1]    DF WL WC GF
Levels:  DF ES F FG GF HW LP PP SF WC WL WP

> ufc[1:5, c(3,5)]                        # first 5 species and heights

  species height
1              NA
2      DF     205
3      WL     330
4      WC      NA
5      GF     300

> ufc[1:5, c("species","height")]         # first 5 species and heights again

  species height
1              NA
2      DF     205
3      WL     330
4      WC      NA
5      GF     300
```

```
> table(ufc$species)

     DF  ES   F  FG  GF  HW  LP  PP  SF  WC  WL  WP
 10  77   3   1   2 185   5   7   4  14 251  34  44
```

The last command suggests that there are ten trees with blank species. Here, it's vital to know about the data recording protocol. The ten trees without species are blank lines to represent empty plots. Let's remove them, for the moment. Note that we also redefine the species factor, so that it will drop the now empty level.

```
> ufc <- ufc[ufc$species != "", ]
> ufc$species <- factor(ufc$species)
```

We can also create new variables within a dataframe, by naming them and assigning them a value. Thus,

```
> ufc$dbh.cm <- ufc$dbh/10
> ufc$height.m <- ufc$height/10
```

Finally, if we want to construct a dataframe from already existing variables, which is quite common, we use the `data.frame()` command, *viz*:

```
> temp <- data.frame(my.species = ufc$species, my.dbh = ufc$dbh.cm)
> temp[1:5, ]

  my.species my.dbh
1         DF     39
2         WL     48
3         WC     15
4         GF     52
5         WC     31
```

Dataframes are the most useful data structures as far as we are concerned. We can use logical vectors that refer to one aspect of the dataframe to extract information from the rest of the dataframe, or even another dataframe. And, it's possible to extract pretty much any information using the tools that we've already seen.

```
> ufc$height.m[ufc$species=="LP"]          # Heights of lodgepole pine

[1] 24.5   NA   NA   NA 16.0 25.0   NA

> mean(ufc$height.m[ufc$species=="LP"], na.rm=TRUE)

[1] 21.83333
```

Sapply

`sapply()` permits us to deploy a function upon each column in a dataframe. This is particularly useful if we want to find column sums or means, for example, but has other useful applications.

```
> sapply(ufc[, 4:7], mean, na.rm = TRUE)

      dbh    height    dbh.cm  height.m
356.56619 240.66496  35.65662  24.06650
```

Below, for example, we use `sapply()` to tell us the class of each of the columns in our dataframe.

```
> sapply(ufc, class)

     plot      tree   species       dbh    height    dbh.cm  height.m
"integer" "integer"  "factor" "integer" "integer" "numeric" "numeric"
```

Exercise 4 Getting to Grips with Data

1. In order to complete the following exercise, be sure that you have executed all the code in the preceding section (starting at page 11).
2. Now, let's try something a little more involved. How would we ask: what are the species of the three tallest trees? This command is best constructed piecemeal. Notice that we are able to nest the subscripts. R starts at the left and works its way right.
   1. `order(ufc$height.m, decreasing = TRUE)` provides the indices of the observations in order of decreasing height.
   2. `ufc$species[order(ufc$height.m, decreasing = TRUE)]` provides the species corresponding to those heights.
   3. `ufc$species[order(ufc$height.m, decreasing = TRUE)][1:3]` provides only the first three.

   ```
   > ufc$species[order(ufc$height.m, decreasing = TRUE)][1:3]

   [1] WP GF WL
   Levels: DF ES F FG GF HW LP PP SF WC WL WP
   ```

   Now, what are the species of the five *fattest* trees?
3. The next useful command is called `tapply()`. This lovely little function allows us to vectorize the application of certain functions to (non-empty) groups of data. In conjunction with factors, this makes for some exceptionally efficient code. `tapply()` requires three things: the target vector to which the function will be applied, the vector by which the target vector will be grouped, and the function to be applied. Any other arguments that you want to pass to function are appended to the call to `tapply()`. That is,

   ```
   > tapply(<variable>, <group>, <function>, <function.arg.2>, ...)
   ```

   Note that the `<variable>` will be used as `<function.arg.1>`.

```
> tapply(ufc$height.m, ufc$species, mean)
```

```
  DF   ES    F   FG   GF   HW   LP   PP   SF   WC   WL   WP
  NA   NA 27.0 27.5   NA 19.8   NA   NA   NA   NA   NA   NA
```

Ah. Many heights are missing, causing the mean to report itself as missing also. Let's fix that by telling the mean to ignore the missing values, temporarily.

```
> tapply(ufc$height.m, ufc$species, mean, na.rm = TRUE)
```

```
      DF       ES        F       FG       GF       HW       LP
25.30000 28.00000 27.00000 27.50000 24.26522 19.80000 21.83333
      PP       SF       WC       WL       WP
33.00000 15.41000 23.48777 25.47273 25.13939
```

And let's pretty it up a little.

```
> format(tapply(ufc$height.m, ufc$species, mean, na.rm = TRUE),
+     dig = 3)
```

```
    DF     ES      F     FG     GF     HW     LP     PP     SF     WC
"25.3" "28.0" "27.0" "27.5" "24.3" "19.8" "21.8" "33.0" "15.4" "23.5"
    WL     WP
"25.5" "25.1"
```

4. What are the mean diameters by species?
5. What are the two species that have the largest third quartile diameters?[2]
6. What are the two species with the largest median slenderness (height/diameter) ratios? How about the two species with the smallest median slenderness ratios?
7. What are the two trees with the largest slenderness ratios? How about two trees with the smallest slenderness ratios?
8. What is the slenderness ratio of the tallest tree? The fattest?
9. Let's try something more involved still. How would we pull out the identity of the median height tree of the species that was eighth tallest on average? That is, identify the tree (if it exists) that has the median height among all trees of the species that has the eighth highest mean height?
   Ok that is ridiculous, but let's stretch the language. Again, it's easiest to work and check our working piecemeal.
   Note that placing the brackets around a statement is short-hand for: "compute this and then print it."
   1. First get the mean height by species.

      ```
      > (ht.bar.by.species <- tapply(ufc$height.m, ufc$species,
      +     mean, na.rm = TRUE))
      ```

      ```
            DF       ES        F       FG       GF       HW       LP
      25.30000 28.00000 27.00000 27.50000 24.26522 19.80000 21.83333
            PP       SF       WC       WL       WP
      33.00000 15.41000 23.48777 25.47273 25.13939
      ```

   2. then get the index order of the species, sorted by average height.

[2] Use the tools that you have already seen in this workshop to figure out what you need to do to compute quartiles.

```
> (species.order.by.ht <- order(ht.bar.by.species, decreasing = TRUE))
 [1]  8  2  4  3 11  1 12  5 10  7  6  9
```

3. The species names in order of the average height by species are then:

```
> (species.by.ht <- levels(ufc$species)[species.order.by.ht])
 [1] "PP" "ES" "FG" "F"  "WL" "DF" "WP" "GF" "WC" "LP" "HW" "SF"
```

4. The eighth tallest is

```
> (sp.8 <- species.by.ht[8])
[1] "GF"
```

5. The median of the heights of all the trees of that species is then

```
> (m.ht.8 <- median(ufc$height.m[ufc$species == sp.8],
+     na.rm = TRUE))
[1] 24.1
```

6. The tree of that species with that height is

```
> ufc[which(ufc$height.m == m.ht.8 & ufc$species == sp.8),
+     ]
    plot tree species dbh height dbh.cm height.m
480   97    3      GF 338    241   33.8     24.1
```

These things must be tackled strategically. If we pursue a rigorous naming policy then these intermediate objects become vital debugging tools as well.
Of course, this can all be expressed as a single operation:

```
> ufc[which(ufc$height.m==median(ufc$height.m[ufc$species ==
+   levels(ufc$species)[order(tapply(ufc$height.m, ufc$species, mean,
+   na.rm = TRUE), decreasing = TRUE)][8]], na.rm = TRUE) &
+   ufc$species==levels(ufc$species)[order(tapply(ufc$height.m,
+   ufc$species, mean, na.rm = TRUE), decreasing = TRUE)][8]),]
    plot tree species dbh height dbh.cm height.m
480   97    3      GF 338    241   33.8     24.1
```

But that would be absurd.

10. What is the identity of the tallest tree of the species that was the fattest on average?

1.3.3 Matrix (Array)

A matrix is simply a vector that has extra attributes, called dimensions. R provides specific algorithms to enable us to treat the vector as though it were really two-dimensional. Many useful matrix operations are available.

```
> (mat.1 <- matrix(c(1,0,1,1), nrow=2))

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

```
> (mat.2 <- matrix(c(1,1,0,1), nrow=2))

     [,1] [,2]
[1,]    1    0
[2,]    1    1

> solve(mat.1)    # This inverts the matrix

     [,1] [,2]
[1,]    1   -1
[2,]    0    1

> mat.1 %*% mat.2 # Matrix multiplication

     [,1] [,2]
[1,]    2    1
[2,]    1    1

> mat.1 + mat.2   # Matrix addition

     [,1] [,2]
[1,]    2    1
[2,]    1    2

> t(mat.1)        # Matrix transposition

     [,1] [,2]
[1,]    1    0
[2,]    1    1

> det(mat.1)      # Matrix determinant

[1] 1
```

There are also various functions of matrices, for example, `qr()` produces the QR decomposition, `eigen()` produces the eigenvalues and eigenvectors of matrices, and `svd()` performs singular value decomposition.

Arrays are one, two, or three-dimensional matrices.

Apply

Apply permits us to deploy a function upon each row or column in a matrix or array. This is particularly useful if we want to find row or column sums or means, for example.

```
> apply(ufc[, 4:7], 2, mean, na.rm = TRUE)

      dbh    height    dbh.cm  height.m
356.56619 240.66496  35.65662  24.06650
```

1.3.4 List

A list is a container for other objects. Lists are invaluable for, for example, collecting and storing complicated output of functions. Lists become invaluable devices as we become more comfortable with R, and start to think of different ways to solve our problems. We access the elements of a list using the double bracket, as below.

```
> (my.list <- list("one", TRUE, 3))

[[1]]
[1] "one"

[[2]]
[1] TRUE

[[3]]
[1] 3

> my.list[[2]]

[1] TRUE
```

If we use a single bracket then we extract the element, still wrapped in the list infrastructure.

```
> my.list[2]

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

We can also name the elements of the list, either during its construction or post-hoc.

```
> (my.list <- list(first = "one", second = TRUE, third = 3))

$first
[1] "one"

$second
[1] TRUE

$third
[1] 3

> names(my.list)

[1] "first"  "second" "third"

> my.list$second

[1] TRUE
```

```
> names(my.list) <- c("First element", "Second element",
+     "Third element")
> my.list

$`First element`
[1] "one"

$`Second element`
[1] TRUE

$`Third element`
[1] 3

> my.list$`Second element`

[1] TRUE
```

Note the deployment of backticks to print the nominated element of the list, even though the name includes spaces.

The output of many functions is a list object. For example, when we fit a least squares regression, the regression object itself is a list, and can be manipulated using list operations.

Above, we saw the use of `tapply()`, which conveniently allowed us to apply an arbitrary function to all the elements of a grouped vector, group by group. We can use `lapply()` to apply an arbitrary function to all the elements of a list.

1.4 Merging Data

It is often necessary to merge datasets that contain data that occupy different hierarchical scales; for example, in forestry we might have some species-level parameters that we wish to merge with tree-level measures in order to make predictions from a known model. We tackled exactly this problem in writing the function that computes the bard-feet volume of a tree given its species, diameter and height. Let's see how this works on a smaller scale. First, we declare the dataframe that contains the species-level parameters.

Recall that by adding the parentheses we are asking R to print the result as well as saving it.

```
> (params <- data.frame(species = c("WP", "WL"),
+                       b0 = c(32.516, 85.150),
+                       b1 = c(0.01181, 0.00841)))

  species     b0      b1
1      WP 32.516 0.01181
2      WL 85.150 0.00841
```

Then let's grab the first three trees of either species from ufc that have non-missing heights. We'll only keep the relevant columns. Note that we can "stack" the index calls, and that they are evaluated sequentially from left to right.

```
> (trees <- ufc[ufc$species %in% params$species & !is.na(ufc$height.m),
+     ][1:3, ])
```

```
   plot tree species dbh height dbh.cm height.m
3     2    2      WL 480    330   48.0       33
20    4    9      WP 299    240   29.9       24
26    5    6      WP 155    140   15.5       14
```

Now we merge the parameters with the species.

```
> (trees <- merge(trees, params))
```

```
  species plot tree dbh height dbh.cm height.m     b0      b1
1      WL    2    2 480    330   48.0       33 85.150 0.00841
2      WP    4    9 299    240   29.9       24 32.516 0.01181
3      WP    5    6 155    140   15.5       14 32.516 0.01181
```

There are many options for merging, for example, in how to deal with not-quite overlapping characteristics, or column names that do not match.

We can now compute the volume using a vectorized approach, which is substantially more efficient than using a loop. Also, note the use of `with()` to *temporarily* attach the dataframe to our search path. This usage simplifies the code.

```
> (trees$volume <- with(trees, b0 + b1 * (dbh.cm/2.54)^2 *
+     (height.m * 3.281)) * 0.002359737)
```

```
[1] 0.9682839 0.3808219 0.1243991
```

1.5 Reshaping Data

Longitudinal datasets often come in the wrong shape for analysis. When longitudinal data are kept in a spreadsheet, it is convenient to devote a row to each object being measured, and include a column for each time point, so that adding new measurements merely requires adding a column. From the point of view of data analysis, however, it is easier to think of each observation on the object as being a row. R makes this switch convenient. Again, we'll see how this works on a small scale.

```
> (trees <- data.frame(tree = c(1, 2), species = c("WH", "WL"),
+                      dbh.1 = c(45, 52), dbh.2 = c(50, 55),
+                      ht.1 = c(30, 35), ht.2 = c(32, 36)))
```

```
  tree species dbh.1 dbh.2 ht.1 ht.2
1    1      WH    45    50   30   32
2    2      WL    52    55   35   36
```

```
> (trees.long <- reshape(trees,
+                        direction = "long",
+                        varying = list(c("dbh.1","dbh.2"),
+                                       c("ht.1","ht.2")),
+                        v.names = c("dbh","height"),
+                        timevar = "time",
+                        idvar = "tree"
+                        ))
```

```
    tree species time dbh height
1.1    1      WH    1  45     30
2.1    2      WL    1  52     35
1.2    1      WH    2  50     32
2.2    2      WL    2  55     36
```

The arguments are defined as follows:

direction tells R to go *wide* or go *long*,

varying is a list of vectors of column names that are to be stacked,

v.names is a vector of the new names for the stacked columns,

timevar is the name of the new column that differentiates between successive measurements on each object, and

idvar is the name of the existing column that differentiates between the objects.

1.6 Sorting Data

R provides the ability to order the elements of objects. Sorting is vital for resolving merging operations. Here are the top five trees by height.

```
> ufc[order(ufc$height.m, decreasing=TRUE),][1:5,]
```

```
    plot tree species  dbh height dbh.cm height.m
413   78    3      WP 1030    480  103.0     48.0
532  110    4      GF  812    470   81.2     47.0
457   88    3      WL  708    425   70.8     42.5
297   55    2      DF  998    420   99.8     42.0
378   68    1      GF  780    420   78.0     42.0
```

`order()` allows ordering by more than one vector, so, for example, to order by plot then species then height, we would use

```
> ufc[order(ufc$plot, ufc$species, ufc$height.m),][1:5,]
```

```
   plot tree species dbh height dbh.cm height.m
2     2    1      DF 390    205     39     20.5
3     2    2      WL 480    330     48     33.0
5     3    2      GF 520    300     52     30.0
8     3    5      WC 360    207     36     20.7
11    3    8      WC 380    225     38     22.5
```

Related functions include `sort()` and `rank()`, but they only permit the use of one index.

Chapter 2

Hierarchical Models

We now shift to the analysis of hierarchical data using mixed-effects models. These models are a natural match for many problems that occur commonly in natural resources.

2.1 Introduction

Recall that for fitting a linear regression using the ordinary techniques that you might be familiar with, you were required to make some assumptions about the nature of the residuals. Specifically, it was necessary to assume that the residuals were

1. independent
2. identically distributed, and, more often than not,
3. normally distributed.

The assumption of constant variance (homoscedasticity) lives in the identically distributed assumption (point 2, above). If these assumptions are true, or even defensible, then life is fine. However, more often than not, we know they're not true. This can happen in natural resources data collections because the data may have a temporal structure, a spatial structure, or a hierarchical structure, or all three[1]. That structure may or may not be relevant to your scientific question, but it's very relevant to the data analysis and modelling!

I mention several references. None are mandatory to purchase or read, but all will be useful at some point or other. They are mentioned in *approximate* order of decreasing utility *for this level*.

2.1.1 Methodological

Pinheiro and Bates (2000) details model fitting in R and Splus, which both provide first-rate graphical model discrimination and assessment tools through the libraries written by the authors. Good examples are provided. Schabenberger and Pierce (2002) is a treasure-trove of sensible advice on understanding and fitting the generalized and mixed-effects models that form the core of this class. There are copious worked examples and there is plenty of SAS code. You are welcome to interact with the material however you see

[1] "The first law of ecology is that everything is related to everything else." – Barry Commoner, US biologist/environmentalist. The Closing Circle: Nature, Man, and Technology. New York : Knopf, 1971.

fit. Finally, Fitzmaurice et al. (2004) and Gelman and Hill (2007) do a first-rate job of explaining the practice of fitting mixed-effects models.

In addition to these books, there are numerous articles that try to explain various elements of these topics in greater or lesser detail. In the past I have found Robinson (1991) (no relation!) and the discussions that follow it particularly useful.

2.1.2 General

Venables and Ripley (2002) is a must-have if you're interested in working with R or Splus. The three previous editions are now legendary in the R/S community for their thorough explication of modern statistical practice, with generous examples and code. The R community has also generated some excellent start-up documents. These are freely available for download at the R project website: `http://www.r-project.org`. Download and read any or all of them, writing code all the way. If you're interested in a deeper exploration of the programming possibilities in R or S then Venables and Ripley (2000) is very useful. Some larger-scale projects that I have been involved with have required calling C programs from R; this reference was very helpful then.

2.2 Some Theory

Mixed effects models contain both fixed and random effects. The model structure is usually suggested by the underlying design or structure of the data. I like to claim that random effects are suggested by the design of a study, and fixed effects are suggested by the hypotheses, but this is not always true.

2.2.1 Effects

"Effects" are predictor variables in a linear or non-linear model[2]. The discussion of fixed and random *effects* can get a little confusing. "Random" and "fixed" aren't normally held to be opposite to one another, or even mutually exclusive (except by sheer force of habit!). Why not "stochastic" and "deterministic"? Or, "sample" and "population"? Or, "local" and "global"? These labels might tell more of the story. Gelman and Hill (2007) decline to use random and fixed altogether.

There are different ways to look at these two properties. Unfortunately, it does affect the data analysis and the conclusions that can be drawn. Modellers may disagree on whether effects should be fixed or random, and the same effect can switch depending on circumstances. Certainly, statisticians haven't agreed on a strategy. Some will claim that it depends entirely on the inference, and some that it depends entirely on the design.

As the statistical tools that are used to analyze such data become more sophisticated, and models previously unthinkable become mainstream, the inadequacies of old vocabularies are increasingly obvious.

Random effects

Random effects are those whose levels are supposedly sampled randomly from a range of possible levels. Generally, although not always, when random effects are considered it is of

[2]The use of the label "effects" is a hang-over from experimental design, and no longer really suits the application, but that's how inertia goes.

interest to connect the results to the broader population. That is, the levels are assumed to be collectively representative of a broader class of potential levels, about which we wish to say something. Alternatively, one might say that a random effect is simply one for which the estimates of location are not of primary interest. Another alternative is that one might say that a random effect is one that you wish to marginalize, for whatever reason.

Fixed effects

Fixed effects are generally assumed to be purposively selected, and represent nothing other than themselves. If an experiment were to be repeated, and the exact same levels of an experimental effect were purposively produced, then the effect is fixed. However, some effects which might vary upon reapplication may also be fixed, so this is not definitive. Alternatively, one might say that a fixed effect is simply one for which the estimates of location are of first interest. Another alternative is that one might say that a fixed effect is one that you wish to condition on, for whatever reason.

Mixed-up effects

Some variables do not lend themselves to easy classification, and either knowledge of process and/or an epistemological slant is required. These are common in natural resources. For example, if an experiment that we feel is likely to be affected by climate is repeated over a number of years, would *year* be a fixed or a random effect? It is not a random sample of possible years, but the same years would not recur if the experiment were repeated. Likewise the replication of an experiment at known locations: some would claim that these should be a fixed effect, others that they represent environmental variation, and therefore they can be considered a random effect.

2.2.2 Model Construction

The process of model construction becomes much more complex now. We have to balance different approaches and assumptions, each of which carries different implications for the model and its utility. If we think about the process of fitting an ordinary regression as being like a flow chart, then adding random effects adds a new dimension to the flow chart altogether. Therefore it's very important to plan the approach before beginning.

The number of potential strategies is as varied as the number of models we can fit. Here is one that we will rely on in our further examples.

1. Choose the minimal set of fixed and random effects for the model.

   (a) Choose the fixed effects that must be there. These effects should be such that, if they are not in the model, the model has no meaning.

   (b) Choose the random effects that must be there. These effects should be such that if they are not in the model, then the model will not adequately reflect the design.

   This is the baseline model, to which others will be compared.

2. Fit this model to the data using tools yet to be discussed, and check the assumption diagnostics. Iterate through the process of improving the random effects, including:

(a) a heteroskedastic variance structure (several candidates)

(b) a correlation structure (several candidates)

(c) extra random effects (e.g. random slopes)

3. When the diagnostics suggest that the fit is reasonable, consider adding more fixed effects. At each step, re-examine the diagnostics to be sure that any estimates that you will use to assess the fixed effects are based on a good match between the data, model, and assumptions.

A further layer of complexity is that it may well be that the assumptions will not be met in the absence of certain fixed effects or random effects. In this case, a certain amount of iteration is inevitable.

It is important to keep in mind that the roles of the fixed and the random effects are distinct. Fixed effects *explain* variation. Random effects *organize* unexplained variation. At the end of the day you will have a model that surficially seems worse than an simple linear regression, by most metrics of model quality. Our goal is to find a model/assumption combination that matches the diagnostics that we examine. Adding random effects adds information, and improves diagnostic compatibility, but explains no more variation!

The bottom line is that the goal of the analyst is to find the simplest model that satisfies the necessary regression assumptions and answers the questions of interest. It is tempting to go hunting for more complex random effects structures, which may provide a higher maximum likelihood, but if the simple model satisfies the assumptions and answers the questions then maximizing the likelihood further is a mathematical exercise - not a statistical one.

Example

In order to illuminate some of these questions, consider the Grand fir stem analysis data. These data are plotted in Figures 2.1 and 2.2.

```
> rm(list = ls())
> stage <- read.csv("../data/stage.csv")
> stage$Tree.ID <- factor(stage$Tree.ID)
> stage$Forest.ID <- factor(stage$Forest, labels = c("Kaniksu",
+     "Coeur d'Alene", "St. Joe", "Clearwater", "Nez Perce",
+     "Clark Fork", "Umatilla", "Wallowa", "Payette"))
> stage$HabType.ID <- factor(stage$HabType, labels = c("Ts/Pach",
+     "Ts/Op", "Th/Pach", "AG/Pach", "PA/Pach"))
> stage$dbhib.cm <- stage$Dbhib * 2.54
> stage$height.m <- stage$Height/3.2808399
```

These habitat codes refer to the climax tree species, which is the most shade-tolerant species that can grow on the site, and the dominant understorey plant, respectively. Ts refers to *Thuja plicata* and *Tsuga heterophylla*, Th refers to just *Thuja plicata*, AG is *Abies grandis*, PA is *Picea engelmanii* and *Abies lasiocarpa*, Pach is *Pachistima myrsinites*, and Op is the nasty *Oplopanaz horridurn*. Grand fir is considered a major climax species for AG/Pach, a major seral species for Th/Pach and PA/Pach, and a minor seral species for Ts/Pach and Ts/Op. Loosely speaking, a community is *seral* if there is evidence that at least some of the species are temporary, and *climax* if the community is self-regenerating Daubenmire (1952).

```
> opar <- par(las = 1)
> plot(stage$dbhib.cm, stage$height.m, xlab = "Dbhib (cm)",
+     ylab = "Height (m)")
> par(opar)
```

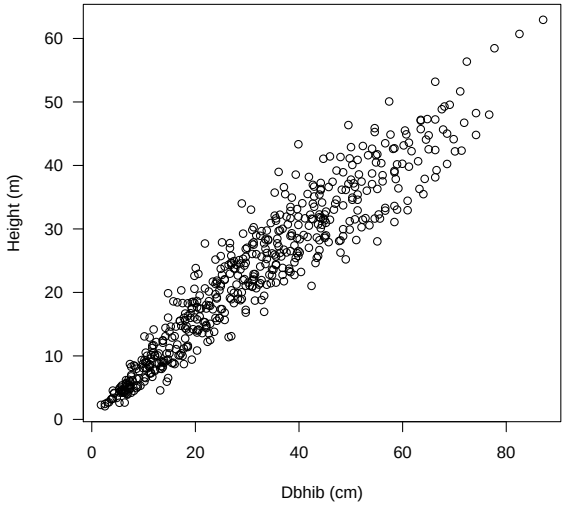

Figure 2.1: Al Stage's Grand Fir stem analysis data: height (ft) against diameter (in). These were dominant and co-dominant trees.

An earlier version of this document required a reasonably fussy nested loop to produce Figure 2.2, which I include below for reference, and portability.

```
> colours <- c("deepskyblue","goldenrod","purple",
+              "orangered2","seagreen")
> par(mfrow=c(3,3), pty="m", mar=c(2, 2, 3, 1) + 0.1, las=1)
> for (i in 1:length(levels(stage$Forest.ID))) {
+   thisForest <- levels(stage$Forest.ID)[i]
+   forestData <- stage[stage$Forest.ID==thisForest,]
+   plot(stage$dbhib.cm, stage$height.m, xlab = "", ylab = "",
+        main = thisForest, type="n")
+   theseTrees <- factor(forestData$Tree.ID)
+   legend("topleft",
+          unique(as.character(forestData$HabType.ID)),
+          xjust=0, yjust=1, bty="n",
+          col=colours[unique(forestData$HabType)],
+          lty=unique(forestData$HabType)+1)
+   for (j in 1:length(levels(theseTrees))) {
+     thisTree <- levels(theseTrees)[j]
+     lines(forestData$dbhib.cm[forestData$Tree.ID==thisTree],
+           forestData$height.m[forestData$Tree.ID==thisTree],
+           col=colours[forestData$HabType[forestData$Tree.ID==thisTree]],
+           lty=forestData$HabType[forestData$Tree.ID==thisTree]+1)
```

```
+   }
+ }
> mtext("Height (m)", outer=T, side=2, line=2)
> mtext("Diameter (cm)", outer=T, side=1, line=2)
```

However, application of Hadley Wickham's powerful ggplot2 package simplifies this challenge. Thanks are due to Charlotte Wickham for the code.

```
> require(ggplot2)

> qplot(dbhib.cm, height.m,
+       data = stage, group = Tree.ID,
+       geom = "line",
+       facets = ~ Forest.ID,
+       colour = HabType.ID,
+       linetype = HabType.ID,
+       xlab = "Diameter (cm)", ylab = "Height (m)"
+       ) +
+   scale_colour_manual(name = "Habitat Type",
+                       values = c("deepskyblue","goldenrod","purple",
+                         "orangered2","seagreen")) +
+   scale_linetype_manual(name = "Habitat Type", values = c(1,2,4:6))
```

Exercise 5 Stage by Stage

Consider the problem of predicting tree height from diameter using the Stage data. There are several candidate predictor variables that could explain variation in the relationship. For example, we might expect the slenderness (height/diameter) relation to differ between trees, between habitat types, and between forests. Which of these different possible effects should be *fixed*, and which *random*? Think about the issues outlined above.

2.2.3 Dilemma

An easy way to approach the advantages to modelling that are offered by mixed-effects models is to think about a simple example. Imagine that we are interested in constructing a height-diameter relationship using two randomly selected plots in a forest, and that we have measured three trees on each. It turns out that the growing conditions are quite different on the plots, leading to a systematic difference between the height–diameter relationship on each (Figure 2.3).

The model is:

$$y_i = \beta_0 + \beta_1 \times x_i + \epsilon_i \tag{2.1}$$

where β_0 and β_1 are fixed but unknown population parameters, and ϵ_i are residuals. The following assumptions are required:

- The true relationship is linear.
- $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

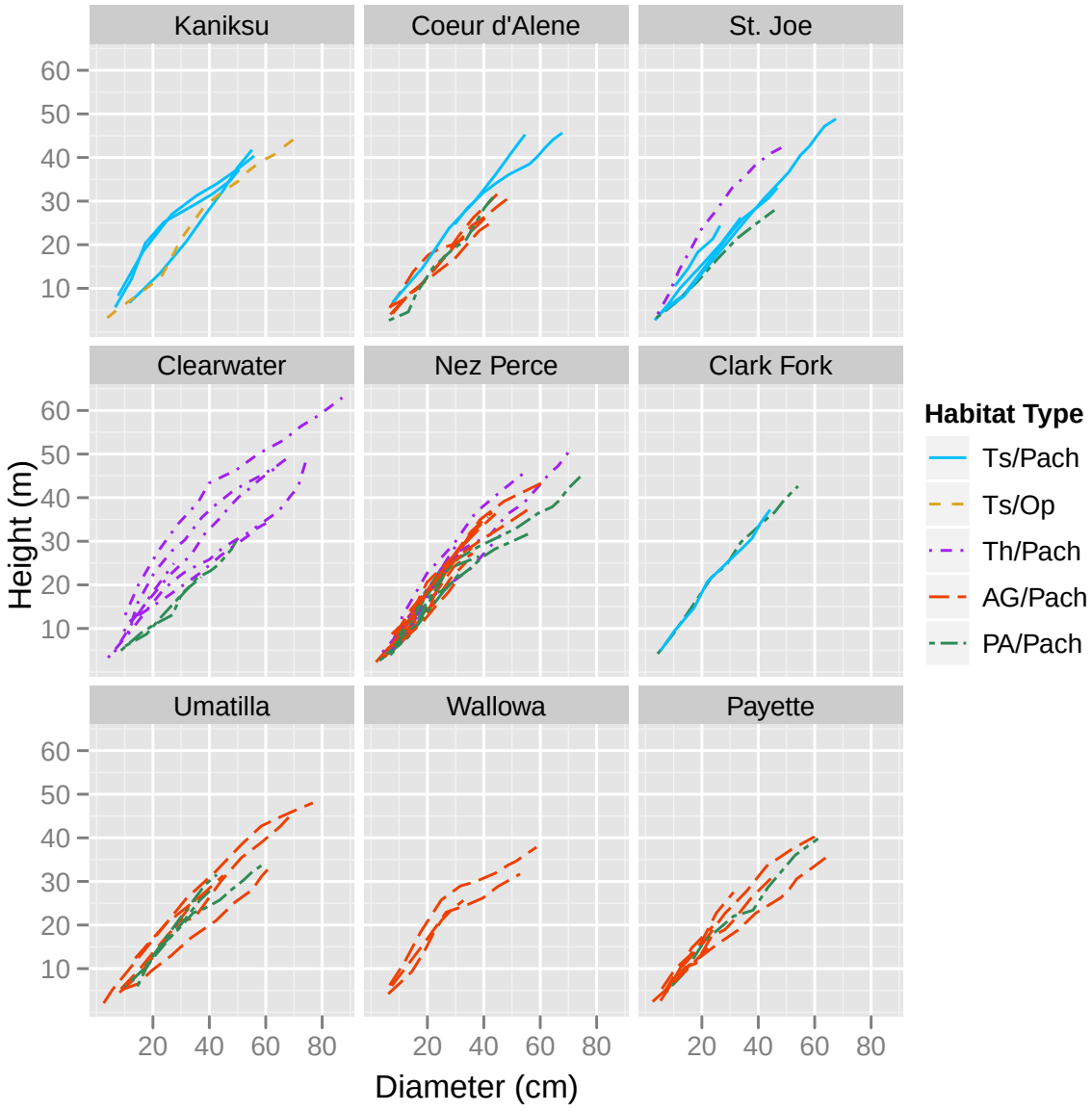

Figure 2.2: Al Stage's Grand Fir Stem Analysis Data: height (ft, vertical axes) against diameter (inches, horizontal axes) by national forest. These were dominant and co-dominant trees.

- The ϵ_i are independent.

```
> trees <- data.frame(plot=factor(c(1, 1, 1, 2, 2, 2)),
+                     dbh.cm=c(30, 32, 35, 30, 33, 35),
+                     ht.m=c(25, 30, 40, 30, 40, 50))

> plot(trees$dbh.cm, trees$ht.m, pch = c(1, 19)[trees$plot],
+     xlim = c(29, 36), xlab = "Diameter (cm)", ylab = "Height (m)")
> abline(lm(ht.m ~ dbh.cm, data = trees), col = "darkgrey")
```

If we fit a simple regression to the trees then we obtain a residual/fitted value plot as displayed in Figure 2.4).

```
> case.model.1 <- lm(ht.m ~ dbh.cm, data = trees)
> plot(fitted(case.model.1), residuals(case.model.1),
+     ylab = "Residuals", xlab = "Fitted Values", pch = c(1,
+         19)[trees$plot])
> abline(h = 0, col = "darkgrey")
```

If we fit a simple regression to the trees with an intercept for each plot then we obtain a residual/fitted value plot as displayed in Figure 2.5.

```
> case.model.2 <- lm(ht.m ~ dbh.cm*plot, data=trees)
> plot(fitted(case.model.2), residuals(case.model.2),
+      xlab="Fitted Values", ylab="Residuals",
+      pch=c(1, 19)[trees$plot])
> abline(h=0, col="darkgrey")
```

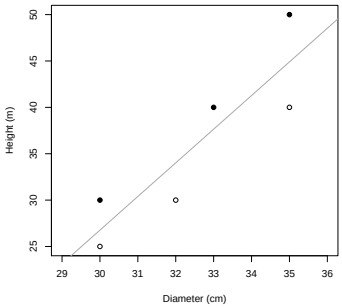

Figure 2.3: Height-diameter measures for three trees on two plots (full and outline symbols), with line of least squares regression.

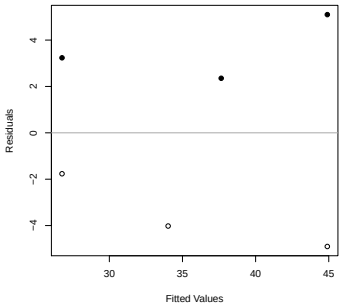

Figure 2.4: Residual plot for height-diameter model for three trees on two plots (full and outline symbols).

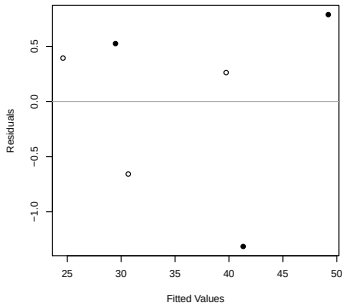

Figure 2.5: Residual plot for height-diameter model including plot for three trees on two plots (full and outline symbols).

These three figures (2.3—2.5) show the analyst's dilemma. The residuals in Figure 2.4 clearly show a correlation within plot; the plot conditions dictate that all the trees in the plot will have the same sign of residual. This phenomenon is not seen in Figure 2.5. However, the model described by Figure 2.5 has no utility, because in order to use it we have to choose whether the tree belongs to plot 1 or plot 2. If the tree belongs to neither, which is true of all of the unmeasured trees, then the model can make no prediction. So, the dilemma is: we can construct a useless model that satisfies the regression assumptions or a useful model that does not.

2.2.4 Decomposition

The dilemma documented in section 2.2.3 has at least two solutions. One is the use of mixed-effects models, the other, which we do not cover here, is explicit modeling of the correlation structure using generalized least squares.

The mixed effects models approach is to decompose the unknown variation into smaller pieces, each of which themselves satisfies the necessary assumptions. Imagine that we

could take the six residual values presented in Figure 2.4, which have the plot-level correlation structure, and decompose them into two plot-level errors and size within-plot errors. That is, instead of:

$$y_{ij} - \hat{y}_{ij} = \hat{\epsilon}_{ij} \tag{2.2}$$

we could try:

$$y_{ij} - \hat{y}_{ij} = \hat{b}_i + \hat{\epsilon}_{ij} \tag{2.3}$$

Then we merely need to assume that:

- The true relationship is linear.
- $b_i \sim \mathcal{N}(0, \sigma_b^2)$
- $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$
- The ϵ_{ij} are independent.

However, when the time comes to use the model for prediction, we do not need to know the plot identity, as the fixed effects do not require it.

This example illustrates the use of random effects. Random effects do not explain variation, that is the role of the fixed effects. Random effects organize variation, or enforce a more complex structure upon it, in such a way that a match is possible between the model assumptions and the diagnostics. In fact, we would expect the overall uncertainty, measured as root mean squared error, to increase any time we fit in any way other than by least squares.

2.3 A Simple Example

We start with a very simple and abstract example. First we have to load the package that holds the mixed-effects code, `nlme`.

```
> require(nlme)
> require(lattice)
```

Now, we generate a simple dataset.

```
> straw <- data.frame(y = c(10.1, 14.9, 15.9, 13.1, 4.2, 4.8, 5.8, 1.2),
+                     x = c(1, 2, 3, 4, 1, 2, 3, 4),
+                     group = factor(c(1, 1, 1, 1, 2, 2, 2, 2)))
```

Let's plot the data (Figure 2.6).

```
> colours = c("red", "blue")
> plot(straw$x, straw$y, col = colours[straw$group])
```

For each model below, examine the output using `summary()` commands of each model, and try to ascertain what the differences are between the models, and whether increasing the complexity seems to be worthwhile. Use `anova()` commands for the latter purpose.

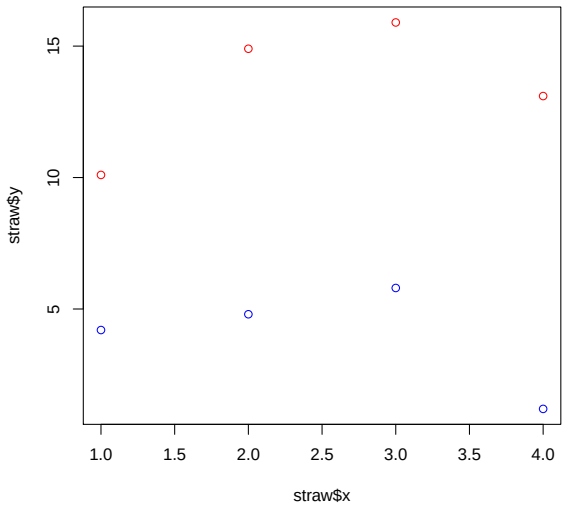

Figure 2.6: A simple dataset to show the use of mixed-effects models.

Ordinary Least Squares

This model is just trying to predict y using x. Using algebra, we would write

$$y_i = \beta_0 + \beta_1 \times x_i + \epsilon_i \tag{2.4}$$

where β_0 and β_1 are fixed but unknown population parameters, and ϵ_i are residuals. The following assumptions are required:

- True relationship is linear.
- $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
- The ϵ_i are independent.

Note that the values of β_0 and β_1 that minimize the residual sum of squares are the least squares estimates in any case, and are unbiased if assumption 1 is true. If assumptions 2 and 3 are true as well then the estimates also have other desirable properties.

The model is fit using R with the following code:

```
> basic.1 <- lm(y ~ x, data = straw)
```

Let's let each `group` have its own intercept. In algebra,

$$y_i = \beta_{01} + \beta_{02} \times g_i + \beta_1 \times x_i + \epsilon_i \tag{2.5}$$

where β_{01}, β_{02}, and β_1 are fixed but unknown population parameters, g_i is an indicator variable with value 0 for group 1 and 1 for group 2, and ϵ_i are residuals. The same assumptions are required as for model 2.4.

The model is fit using R with the following code:

```
> basic.2 <- lm(y ~ x + group, data = straw)
```

Let's let each `group` have its own intercept *and* slope.

$$y_i = \beta_{01} + \beta_{02} \times g_i + (\beta_{11} + \beta_{12} \times g_i) \times x_i + \epsilon_i \tag{2.6}$$

where β_{01}, β_{02}, β_{12}, and β_{12} are fixed but unknown population parameters, g_i is an indicator variable with value 0 for group 1 and 1 for group 2, and ϵ_i are residuals. The same assumptions are required as for model 2.4.

The model is fit using R with the following code:

```
> basic.3 <- lm(y ~ x * group, data = straw)
```

Mixed Effects

Now we need to convert the data to a grouped object - a special kind of dataframe that allows special `nlme()` commands. The `group` will hereby be a random effect. One command that we can now use is `augPred`, as seen below. Try it on a few models.

```
> straw.mixed <- groupedData(y ~ x | group, data = straw)
```

Now let's fit the basic mixed-effects model that allows the intercepts to vary randomly between the groups. We'll add a subscript for clarity.

$$y_{ij} = \beta_0 + b_{0i} + \beta_1 \times x_{ij} + \epsilon_{ij} \tag{2.7}$$

where β_0 and β_1 are fixed but unknown population parameters, the b_{0i} are two group-specific random intercepts, and ϵ_{ij} are residuals. The following assumptions are required:

- True relationship is linear.
- $b_{0i} \sim \mathcal{N}(0, \sigma^2_{b_0})$
- $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$
- The ϵ_{ij} are independent.

The model is fit using R with the following code:

```
> basic.4 <- lme(y ~ x, random = ~1 | group, data = straw.mixed)
```

The `random` syntax can be a little confusing. Here, we're instructing R to let each group have its own random intercept. If we wanted to let each group have its own slope and intercept, we would write `random = x`. If we wanted to let each group have its own slope but not intercept, we would write `random = x - 1`.

We can examine the model in a useful graphic called an *augmented prediction plot*. This plot provides a scatterplot of the data, split up by group, and a fitted line which represents the model predictions (Figure 2.7). We should also check the regression diagnostics that are relevant to our assumptions, but we have so few data here that examples aren't useful. We will develop these ideas further during the case study, to follow.

```
> plot(augPred(basic.4))
```

If we are satisfied with the model diagnostics then we can examine the structure of the model, including the estimates, using the `summary()` function. The summary function presents a collection of useful information about the model. Here we report the default structure for a `summary.lme` object. The structure may change in the future, but the essential information will likely remain the same.

```
> summary(basic.4)
```

Firstly, the `data.frame` object is identified, and fit statistics reported, including Akaike's "An Information Criterion", Schwartz's "Bayesian Information Criterion", and the log likelihood.

```
Linear mixed-effects model fit by REML
 Data: straw.mixed
       AIC      BIC    logLik
  43.74387 42.91091 -17.87193
```

The random effects structure is then described, and estimates are provided for the parameters. Here we have an intercept for each group, and the standard deviation is reported, as well as the standard deviation of the residuals within each group.

```
Random effects:
 Formula: ~1 | group
        (Intercept) Residual
StdDev:    6.600769 2.493992
```

The fixed effects structure is next described, in a standard t-table arrangement. Estimated correlations between the fixed effects follow.

```
Fixed effects: y ~ x
            Value Std.Error DF   t-value p-value
(Intercept)   8.5  5.142963  5 1.6527437  0.1593
x             0.1  0.788670  5 0.1267958  0.9040
 Correlation:
  (Intr)
x -0.383
```

The distribution of the within-group residuals, also called the *innermost residuals* in the context of strictly hierarchical models by Pinheiro and Bates (2000), is then described.

```
Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-1.2484738 -0.4255493  0.1749470  0.6387985  1.0078956
```

Finally the hierarchical structure of the model and data is presented.

```
Number of Observations: 8
Number of Groups: 2
```

Next let's fit a unique variance to each group. The model form will be the same as in equation 2.7, but the assumptions will be different. Now, we will need to assume that

- True relationship is linear.
- $b_{0i} \sim \mathcal{N}(0, \sigma^2_{b_1})$
- $\epsilon_{1j} \sim \mathcal{N}(0, \sigma^2_{b01})$
- $\epsilon_{2j} \sim \mathcal{N}(0, \sigma^2_{b02})$
- $Cov(\epsilon_{ab}, \epsilon_{cd}) = 0$ for $a \neq c$ or $d \neq d$

The model is fit using R with the following code:

```
> basic.5 <- lme(y ~ x, random = ~1 | group, weights = varIdent(form = ~1 |
+     group), data = straw.mixed)
```

The summary output is essentially identical to the previous in structure, with the addition of a new section that summarizes the newly-added variance model. Here we show only the new portion.

```
> summary(basic.5)

Variance function:
 Structure: Different standard deviations per stratum
 Formula: ~1 | group
 Parameter estimates:
       2        1
1.000000 1.327843
```

Finally let's allow for temporal autocorrelation within each group. Again, the model form will be the same as in equation 2.7, but the assumptions will be different. Now, we will need to assume that

- True relationship is linear.
- $b_{0i} \sim \mathcal{N}(0, \sigma^2_{b_1})$
- $\epsilon_{1j} \sim \mathcal{N}(0, \sigma^2_{b01})$
- $\epsilon_{2j} \sim \mathcal{N}(0, \sigma^2_{b02})$
- $Cov(\epsilon_{ab}, \epsilon_{ac}) = \rho$ for $b \neq c$
- The ϵ_{ij} are independent otherwise.

The model is fit using R with the following code:

```
> basic.6 <- lme(y ~ x, random = ~1 | group, weights = varIdent(form = ~1 |
+     group), correlation = corAR1(), data = straw.mixed)
```

The summary output is again essentially identical to the previous in structure, with the addition of a new section that summarizes the newly-added correlation model. Here we show only the new portion.

```
> summary(basic.6)

Correlation Structure: AR(1)
 Formula: ~1 | group
 Parameter estimate(s):
      Phi
0.8107325
```

We can summarize some of these differences in a graph (Figure 2.8).

```
> opar <- par(las=1)
> colours <- c("blue", "darkgreen", "plum")
> plot(straw$x, straw$y)
> for (g in 1:2) lines(straw$x[straw$group == levels(straw$group)[g]],
+                      fitted(basic.1)[straw$group ==
+                                      levels(straw$group)[g]],
+                      col = colours[1])
> for (g in 1:2) lines(straw$x[straw$group == levels(straw$group)[g]],
+                      fitted(basic.2)[straw$group ==
+                                      levels(straw$group)[g]],
+                      col = colours[2])
> for (g in 1:2) lines(straw$x[straw$group == levels(straw$group)[g]],
+                      fitted(basic.4)[straw$group ==
+                                      levels(straw$group)[g]],
+                      col = colours[3])
> legend(2.5, 13, lty = rep(1, 3), col = colours,
+        legend = c("Mean Only", "Intercept Fixed", "Intercept Random"))
> par(opar)
```

2.3.1 The Deep End

There are numerous different representations of the linear mixed-effects model. We'll adopt that suggested by Laird and Ware (1982):

$$\begin{aligned} \mathbf{Y} &= \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b} + \boldsymbol{\epsilon} \\ \mathbf{b} &\sim \mathcal{N}(\mathbf{0}, \mathbf{D}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned}$$

Here, $\mathbf{D}$ and $\mathbf{R}$ are preferably constructed using a small number of parameters, which will be estimated from the data. We'll think first about estimation using maximum likelihood.

2.3.2 Maximum Likelihood

Recall that the principle behind maximum likelihood was to find the suite of parameter estimates that were best supported by the data. This began by writing down the conditional distribution of the observations. For example the *pdf* for a single observation from the normal distribution is:

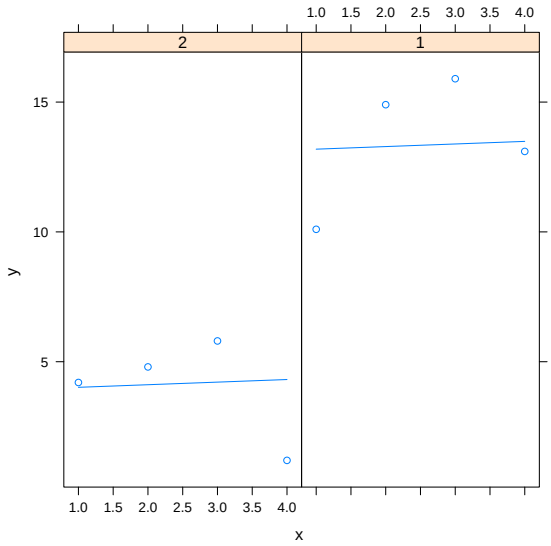

Figure 2.7: An augmented plot of the basic mixed-effects model with random intercepts fit to the sample dataset.

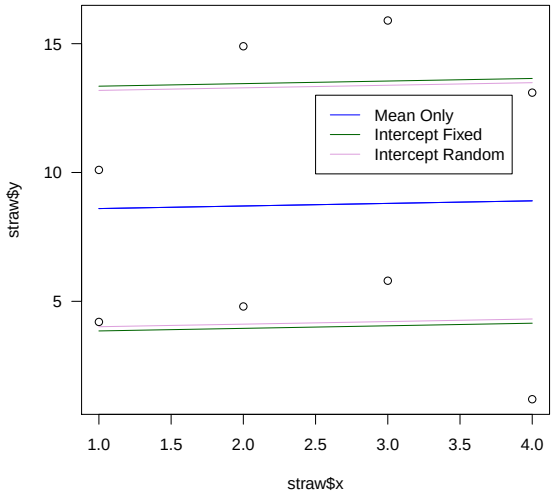

Figure 2.8: A sample plot showing the difference between basic.1 (single line), basic.2 (intercepts are fixed), and basic.4 (intercepts are random).

$$f\left(y_i \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(y_i-\mu)^2}{2\sigma^2}}$$

So if $Y \overset{\mathrm{d}}{=} \mathrm{N}\left(\mu, \mathbf{V}\right)$ then by definition:

$$f\left(\mathbf{Y} \mid \mu, \mathbf{V}\right) = \frac{|\mathbf{V}|^{-\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}(\mathbf{Y}-\mu)'\mathbf{V}^{-1}(\mathbf{Y}-\mu)}$$

So in terms of the linear model $\mathbf{Y} = \mathbf{X}\beta$, the conditional joint density is

$$f\left(\mathbf{Y} \mid \mathbf{X}, \beta, \mathbf{V}\right) = \frac{|\mathbf{V}|^{-\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}(\mathbf{Y}-\mathbf{X}\beta)'\mathbf{V}^{-1}(\mathbf{Y}-\mathbf{X}\beta)}$$

Reversing the conditioning and taking logs yields:

$$\mathcal{L}\left(\beta, \mathbf{V} \mid \mathbf{Y}, \mathbf{X}\right) = -\frac{1}{2}\ln\left(|\mathbf{V}|\right) - \frac{n}{2}\ln\left(2\pi\right) - \frac{1}{2}\left(\mathbf{Y}-\mathbf{X}\beta\right)'\mathbf{V}^{-1}\left(\mathbf{Y}-\mathbf{X}\beta\right)$$

Notice that the parameters we're interested in are now embedded in the likelihood. Solving for those parameters should be no more difficult than maximizing the likelihood. In theory. Now, to find $\hat{\beta}$ we take the derivative of $\mathcal{L}\left(\beta, \mathbf{V} \mid \mathbf{y}, \mathbf{X}\right)$ with regards to β:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\beta} = \frac{\mathrm{d}}{\mathrm{d}\beta}\left[-\frac{1}{2}\left(\mathbf{y}-\mathbf{X}\beta\right)'\mathbf{V}^{-1}\left(\mathbf{y}-\mathbf{X}\beta\right)\right]$$

this leads to, as we've seen earlier

$$\hat{\beta}_{MLE} = \left(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{Y}$$

but this only works *if we know* $\mathbf{V}$!

Otherwise, we have to maximize the likelihood as follows. First, substitute

$$(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{Y}$$

for β in the likelihood. That is, remove all the instances of β, and replace them with this statement. By this means, β is *profiled out* of the likelihood. The likelihood is now only a function of the data and the covariance matrix V. This covariance matrix is itself a function of the covariance matrices of the random effects, which are structures that involve hopefully only a few unknown parameters, and that are organized by the model assumptions.

Maximize the resulting likelihood in order to estimate $\hat{V}$, and then calculate the estimate of the fixed effects via:

$$\hat{\beta}_{MLE} = \left(\mathbf{X}'\hat{\mathbf{V}}^{-1}\mathbf{X}\right)^{-1}\mathbf{X}'\hat{\mathbf{V}}^{-1}\mathbf{Y} \tag{2.8}$$

After some tedious algebra, which is well documented in Schabenberger and Pierce (2002), we also get the $BLUP$s.

$$\hat{b}_{MLE} = \mathbf{D}\mathbf{Z}'\hat{\mathbf{V}}\left(\mathbf{Y} - \mathbf{X}\hat{\beta}\right) \tag{2.9}$$

where $\mathbf{D}$ is the covariance matrix of the random effects.

2.3.3 Restricted Maximum Likelihood

It was noted earlier that maximum likelihood estimators of covariance parameters are usually negatively biased. This is because in profiling out the fixed effects, we're effectively pretending that we know them, and therefore we are not reducing the degrees of freedom appropriately. *Restricted* or *Residual* Maximum Likelihood will penalize the estimation based on the model size, and is therefore a preferred strategy. $ReML$ is not unbiased, except under certain circumstances, but it is less biased than maximum likelihood.

Instead of maximizing the conditional joint likelihood of $\mathbf{Y}$ we do so for an almost arbitrary linear transformation of $\mathbf{Y}$, which we shall denote $\mathbf{K}$. It is almost arbitrary inasmuch as there are only two constraints: $\mathbf{K}$ must have full column rank, or else we would be creating observations out of thin air, and $\mathbf{K}$ must be chosen so that $\mathrm{E}[\mathbf{K}'\mathbf{Y}] = 0$.

The easiest way to guarantee that these hold is to ensure that $[\mathbf{K}'\mathbf{X}] = 0$, and that $\mathbf{K}$ has no more than $n - p$ independent columns, where p is the number of independent parameters in the model. Notice that we would like $\mathbf{K}$ to have as many columns as it can because this will translate to more realizations for fitting the model. This removes the fixed effects from consideration and in so doing also penalizes the estimation for model size. So, the likelihood is restricted by the fixed effects being set to 0, thus, restricted maximum likelihood. Finally, notice that having a $\mathbf{0}$ column in $\mathbf{K}$ doesn't actually add any information to the problem.

So, briefly, $ReML$ involves applying ML, but replacing $\mathbf{Y}$ with $\mathbf{K}\mathbf{Y}$, $\mathbf{X}$ with $\mathbf{0}$, $\mathbf{Z}$ with $\mathbf{K}'\mathbf{Z}$, and $\mathbf{V}$ with $\mathbf{K}'\mathbf{V}\mathbf{K}$.

2.4 Case Study

Recall our brutal exposition of the mixed-effects model:

$$\begin{aligned} \mathbf{Y} &= \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b} + \boldsymbol{\epsilon} \\ \mathbf{b} &\sim \mathcal{N}(\mathbf{0}, \mathbf{D}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned}$$

D and **R** are covariance matrices, constructed using a small number of parameters, and the structure of which is *suggested* by what is known about the data and can be *tested* by comparing nested models.

2.4.1 Stage Data

A brief synopsis: a sample of 66 trees was selected in national forests around northern and central Idaho. According to Stage (*pers. comm.* 2003), the trees were selected purposively rather than randomly. Stage (1963) notes that the selected trees "... appeared to have been dominant throughout their lives" and "... showed no visible evidence of crown damage, forks, broken tops, etc." The habitat type and diameter at 4'6" were also recorded for each tree, as was the national forest from which it came. Each tree was then split, and decadal measures were made of height and diameter inside bark at 4'6".

First, eyeball the data in your spreadsheet of choice. Then import the data as follows:

```
> rm(list = ls())
> stage <- read.csv("../data/stage.csv")
> dim(stage)

[1] 542   7

> names(stage)

[1] "Tree.ID" "Forest"  "HabType" "Decade"  "Dbhib"   "Height"
[7] "Age"

> sapply(stage, class)

  Tree.ID    Forest   HabType    Decade     Dbhib    Height
"integer" "integer" "integer" "integer" "numeric" "numeric"
      Age
"integer"
```

Some cleaning will be necessary. Let's start with the factors.

```
> stage$Tree.ID <- factor(stage$Tree.ID)
> stage$Forest.ID <- factor(stage$Forest, labels = c("Kan",
+     "Cd'A", "StJoe", "Cw", "NP", "CF", "Uma", "Wall",
+     "Ptte"))
> stage$HabType.ID <- factor(stage$HabType, labels = c("Ts/Pach",
+     "Ts/Op", "Th/Pach", "AG/Pach", "PA/Pach"))
```

The measurements are all imperial (this was about 1960, after all).

```
> stage$dbhib.cm <- stage$Dbhib * 2.54
> stage$height.m <- stage$Height/3.2808399
> str(stage)

'data.frame':        542 obs. of  11 variables:
 $ Tree.ID   : Factor w/ 66 levels "1","2","3","4",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ Forest    : int  4 4 4 4 4 4 4 4 4 4 ...
 $ HabType   : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Decade    : int  0 1 2 3 4 0 1 2 3 4 ...
 $ Dbhib     : num  14.6 12.4 8.8 7 4 20 18.8 17 15.9 14 ...
 $ Height    : num  71.4 61.4 40.1 28.6 19.6 ...
 $ Age       : int  55 45 35 25 15 107 97 87 77 67 ...
 $ Forest.ID : Factor w/ 9 levels "Kan","Cd'A","StJoe",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ HabType.ID: Factor w/ 5 levels "Ts/Pach","Ts/Op",..: 5 5 5 5 5 5 5 5 5 5 ...
 $ dbhib.cm  : num  37.1 31.5 22.4 17.8 10.2 ...
 $ height.m  : num  21.76 18.71 12.22 8.72 5.97 ...
```

Height from Diameter

The prediction of height from diameter provides useful and inexpensive information. It may be that the height/diameter relationship differs among habitat types, or climate zones, or tree age. Let's examine the height/diameter model of the trees using a mixed-effects model. We'll start with a simple case, using only the oldest measurement from each tree that provides one.

```
> stage.old <- stage[stage$Decade == 0, ]
```

Note that this code actually drops a tree, but we can afford to let it go for this demonstration.

To establish a baseline of normalcy, let's first fit the model using ordinary least squares. We drop the sole observation from the Ts/Op habitat type. It will cause trouble otherwise (the leverage will prove to be very high).

```
> hd.lm.1 <- lm(height.m ~ dbhib.cm * HabType.ID, data = stage.old,
+     subset = HabType.ID != "Ts/Op")
```

Formally, I think it is good practice to examine the diagnostics upon which the model is predicated *before* examining the model itself, tempting though it may be, so see Figure 2.9. The graph of the residuals vs. fitted values plot (top left) seems good. There is no suggestion of heteroskedasticity. The Normal Q-Q plot suggests a little wiggle but seems reasonably straight. There seem to be no points of egregious influence (bottom left; all Cook's Distances < 1).

```
> opar <- par(mfrow = c(2, 2), mar = c(4, 4, 4, 1))
> plot(hd.lm.1)
> par(opar)
```

So, having come this far, we should examine the model summary.

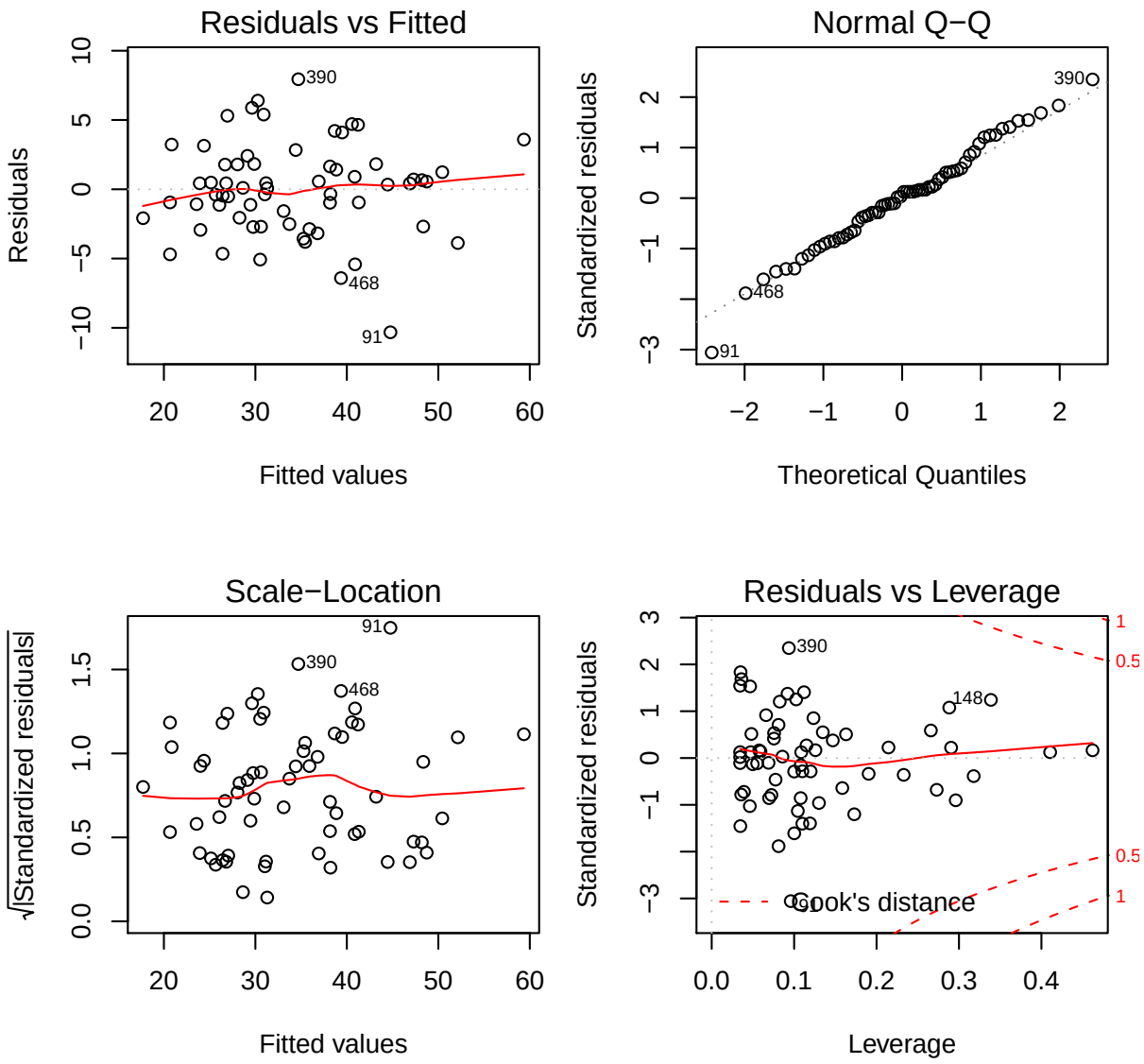

Figure 2.9: Regression diagnostics for the ordinary least squares fit of the Height/Diameter model with habitat type for Stage's data.

```
> summary(hd.lm.1)

Call:
lm(formula = height.m ~ dbhib.cm * HabType.ID, data = stage.old,
    subset = HabType.ID != "Ts/Op")

Residuals:
     Min       1Q   Median       3Q      Max
-10.3210  -2.1942   0.2218   1.7992   7.9437

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                8.33840    4.64118   1.797   0.0778
```

```
dbhib.cm                     0.58995    0.08959   6.585 1.67e-08
HabType.IDTh/Pach            2.42652    5.78392   0.420   0.6764
HabType.IDAG/Pach            0.29582    5.13564   0.058   0.9543
HabType.IDPA/Pach            0.02604    5.96275   0.004   0.9965
dbhib.cm:HabType.IDTh/Pach -0.03224    0.10670  -0.302   0.7637
dbhib.cm:HabType.IDAG/Pach -0.08594    0.10116  -0.850   0.3992
dbhib.cm:HabType.IDPA/Pach -0.10322    0.11794  -0.875   0.3852

(Intercept)                  .
dbhib.cm                     ***
HabType.IDTh/Pach
HabType.IDAG/Pach
HabType.IDPA/Pach
dbhib.cm:HabType.IDTh/Pach
dbhib.cm:HabType.IDAG/Pach
dbhib.cm:HabType.IDPA/Pach
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.551 on 56 degrees of freedom
Multiple R-squared: 0.8748,        Adjusted R-squared: 0.8591
F-statistic: 55.89 on 7 and 56 DF,  p-value: < 2.2e-16
```

For comparison: the following quantities are in metres. The first is the standard deviation of the height measures. The second is the standard deviation of the height measures conditional on the diameter measures *and* the model.

```
> sd(stage.old$height.m)

[1] 9.468042

> summary(hd.lm.1)$sigma

[1] 3.551062
```

It's also interesting to know how much variation is explained by the habitat type information. We can assess this similarly. Here we will not worry about diagnostics, although it should be done.

```
> summary(lm(height.m ~ dbhib.cm, data = stage.old,
+     subset = HabType.ID != "Ts/Op"))$sigma

[1] 4.101350
```

Not much!

Mixed effects

Based on our knowledge of the locations of national forests, it seems reasonable to believe that there will be similarities between trees that grow in the same forest *relative to the population of trees*.

However, we'd like to create a model that doesn't rely on knowing the national forest, that is, a model that can plausibly be used for trees in other forests. This is acceptable as long as we are willing to believe that the sample of trees that we are using is representative of the conditions for which we wish to apply the model. In the absence of other information, this is a judgement call. Let's assume it for the moment.

Then, based on the above information, national forest will be a random effect, and habitat type a fixed effect. That is, we wish to construct a model that can be used for any forest, that might be more accurate if used correctly within a named national forest, and provides unique estimates for habitat type. We can later ask how useful the knowledge of habitat type is, and whether we want to include that in the model.

So, we'll have two random effects: national forest and tree within national forest. We have one baseline fixed effect: diameter at breast height inside bark, with to potential additions: age and habitat type. The lowest-level sampling unit will be the tree, nested within national forest.

It is convenient to provide a basic structure to R. The structure will help R create useful graphical diagnostics later in the analysis. Note that you only need to `require()` a package once per session, but it doesn't hurt to do it more often. Here we will scatter them liberally to remind you what packages you should be using.

```
> require(nlme)
> stage.old <- groupedData(height.m ~ dbhib.cm | Forest.ID,
+     data = stage.old)
```

Now, let's look to our model.

$$y_{ij} = \beta_0 + b_{0i} + \beta_1 \times x_{ij} + \epsilon_{ij} \tag{2.10}$$

y_{ij} is the height of tree j in forest i, x_{ij} is the diameter of the same tree. β_0 and β_1 are fixed but unknown parameters and b_{0i} are the forest-specific random and unknown intercepts. Later we might see if the slope also varies with forest. So, in matrix form,

$$Y = \beta\mathbf{X} + b\mathbf{Z} + \epsilon \tag{2.11}$$

Y is the column of tree heights, X will be the column of diameters, with a matrix of 0s and 1s to allocate the observations to different habitat types, along with a column for the ages, if necessary. β will be a vector of parameter estimates. Z will be a matrix of 0s and 1s to allocate the observations to different forests. b will be a vector of means for the forests and trees within forests. Finally, we'll let $\mathbf{D}$ be a 9×9 identity matrix multiplied by a constant σ_h^2, and $\mathbf{R}$ be a 66×66 identity matrix multiplied by a constant σ^2.

```
> hd.lme.1 <- lme(height.m ~ dbhib.cm, random = ~1 |
+     Forest.ID, data = stage.old)
```

Automatic functions are available to extract and plot the different pieces of the model. I prefer to extract them and choose my own plotting methods. I recommend that you do the same. For the pre-programmed versions see Pinheiro and Bates (2000).

A quick burst of jargon: for hierarchical models there is more than one level of fitted values and residuals. Pinheiro and Bates (2000) adopt the following approach: the outermost residuals and fitted values are conditional only on the fixed effects, the innermost residuals and fitted values are conditional on the fixed and all the random effects, and there are as many levels between these extremes as are necessary. So, in a two-level model like this,

- the outermost residuals are the residuals computed from the outermost fitted values, which are computed from only the fixed effects. Let's refer to them as r_0.

$$r_0 = y_{ij} - \hat{\beta}_0 - \hat{\beta}_1 \times xij \tag{2.12}$$

- the innermost residuals are the residuals computed from the innermost fitted values, which are computed from the fixed effects and the random effects. Let's refer to them as r_1.

$$r_1 = y_{ij} - \hat{\beta}_0 - \hat{b}_{0i} - \hat{\beta}_1 \times xij \tag{2.13}$$

Furthermore, the mixed-effects apparatus provides us with three kinds of innermost and outermost residuals:

1. *response* residuals, simply the difference between the observation and the prediction;
2. *Pearson* residuals, which are the response residuals scaled by dividing by their standard deviation; and
3. *normalized* residuals, which are the Pearson residuals pre-multiplied by the inverse square-root of the estimated correlation matrix from the model.

The key assumptions that we're making for our model are that:

1. the model structure is correctly specified;
2. the random effects are normally distributed;
3. the innermost residuals are normally distributed;
4. the innermost residuals are homoscedastic within and across the groups; and
5. the innermost residuals are independent within the groups.

Notice that we're not making any assumptions about the outermost residuals. However, they are useful for summarizing the elements of model performance.

We should construct diagnostic graphs to check these assumptions. Note that in some cases, the assumptions are stated in an untenably broad fashion. Therefore the sensible strategy is to check for the conditions that can be interpreted in the context of the design, the data, and the incumbent model. For example, there are infinite ways that the innermost residuals could fail to have constant variance. What are the important ways? The situation most likely to lead to problems is if the variance of the residuals is a function of something, whether that be a fixed effect or a random effect.

Rather than trust my ability to anticipate what the programmers meant by the labels etc., I want to know what goes into each of my plots. The best way to do that is to put it there myself. To examine each of the assumptions in turn, I have constructed the following suite of graphics. These are presented in Figure 2.10.

1. A plot of the outermost fitted values against the observed values of the response variable. This graph allows an overall summary of the explanatory power of the model.
   (a) How much of the variation is explained?
   (b) How much remains?
   (c) Is there evidence of lack of fit anywhere in particular?
2. A plot of the innermost fitted values against the innermost Pearson residuals. This graph allows a check of the assumption of correct model structure.
   (a) Is there curvature?
   (b) Do the residuals fan out?
3. a qq-plot of the estimated random effects, to check whether they are normally distributed with constant variance.
   (a) Do the points follow a straight line, or do they exhibit skew or kurtosis?
   (b) Are any outliers evident?
4. a qq-plot of the Pearson residuals, to check whether they are normally distributed with constant variance.
   (a) Do the points follow a straight line, or do they exhibit skew or kurtosis?
   (b) Are any outliers evident?
5. a notched boxplot of the innermost Pearson residuals by the grouping variable, to see what the within-group distribution looks like.
   (a) Do the notches intersect 0?
   (b) Is there a trend between the medians of the within-group residuals and the estimated random effect?
6. a scatterplot of the variance of the Pearson residuals within the forest against the forest random effect.
   (a) Is there a distinct positive or negative trend?

NB: I do not tend to use any of the widely-available statistical tests for homoskedasticity, normality, etc, for diagnostics. I like (Box, 1953): "... to make preliminary tests on variances is rather like putting to sea in a rowing boat to find out whether conditions are sufficiently calm for an ocean liner to leave port".

We use the following code to produce Figure 2.10. Of course there is no need to pack all the graphical diagnostics into one figure.

```
> opar <- par(mfrow = c(3, 2), mar = c(4, 4, 3, 1), las = 1, cex.axis = 0.9)
> #### Plot 1
> plot(fitted(hd.lme.1, level=0), stage.old$height.m,
+      xlab = "Fitted Values (height, m.)",
+      ylab = "Observed Values (height, m.)",
```

```
+      main = "Model Structure (I)")
> abline(0, 1, col = "blue")
> #### Plot 2
> scatter.smooth(fitted(hd.lme.1), residuals(hd.lme.1, type="pearson"),
+                xlab = "Fitted Values",
+                ylab = "Innermost Residuals",
+                main = "Model Structure (II)")
> abline(h = 0, col = "red")
> #### Plot 3
> ref.forest <- ranef(hd.lme.1)[[1]]
> ref.var.forest <- tapply(residuals(hd.lme.1, type="pearson", level=1),
+                          stage.old$Forest.ID,  var)
> qqnorm(ref.forest, main="Q-Q Normal - Forest Random Effects")
> qqline(ref.forest, col="red")
> #### Plot 4
> qqnorm(residuals(hd.lme.1, type="pearson"), main="Q-Q Normal - Residuals")
> qqline(residuals(hd.lme.1, type="pearson"), col="red")
> #### Plot 5
> boxplot(residuals(hd.lme.1, type="pearson", level=1) ~ stage.old$Forest.ID,
+         ylab = "Innermost Residuals", xlab = "National Forest",
+         notch=T, varwidth = T, at=rank(ref.forest))
> axis(3, labels=format(ref.forest, dig=2), cex.axis=0.8,
+      at=rank(ref.forest))
> abline(h=0, col="darkgreen")
> #### Plot 6
> plot(ref.forest, ref.var.forest, xlab="Forest Random Effect",
+      ylab="Variance of within-Forest Residuals")
> abline(lm(ref.var.forest ~ ref.forest), col="purple")
> par(opar)
```

Cross-reference these against Figure 2.10. In fact, all of these residual diagnostics look good.

But, let's accept the model as it stands for the moment, and go on to examine the summary. Refer to Section 2.3 for a detailed description of the output that follows.

```
> summary(hd.lme.1)

Linear mixed-effects model fit by REML
 Data: stage.old
       AIC      BIC    logLik
  376.6805 385.2530 -184.3403

Random effects:
 Formula: ~1 | Forest.ID
        (Intercept) Residual
StdDev:    1.151405 3.937486

Fixed effects: height.m ~ dbhib.cm
               Value Std.Error DF   t-value p-value
```

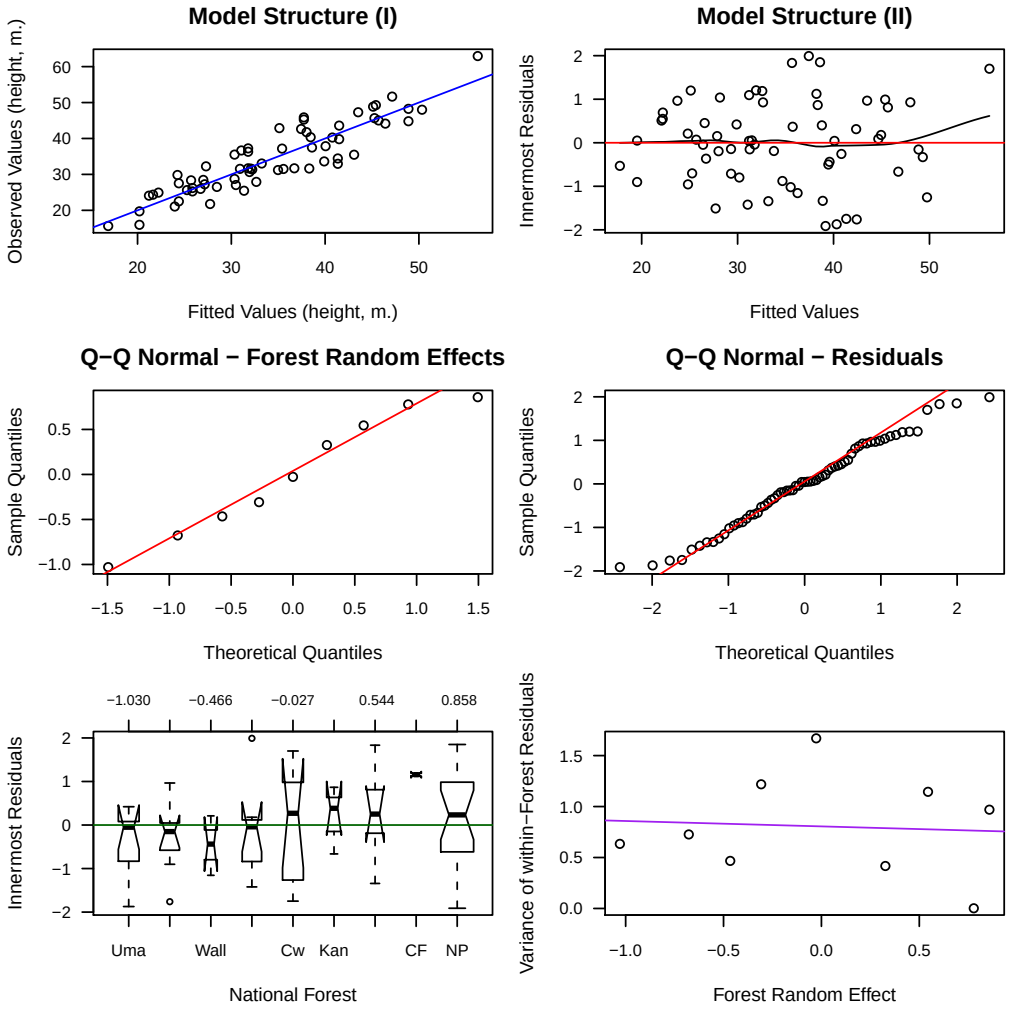

Figure 2.10: Selected diagnostics for the mixed-effects fit of the Height/Diameter ratio against habitat type and national forest for Stage's data.

```
(Intercept) 6.58239 1.7763571 55  3.705556   5e-04
dbhib.cm    0.57036 0.0335347 55 17.008062   0e+00
 Correlation:
            (Intr)
dbhib.cm -0.931

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-1.91215622 -0.70233393  0.04308139  0.81189065  1.99133843

Number of Observations: 65
Number of Groups: 9
```

1. Here we have the overall metrics of model fit, including the log likelihood (recall that this is the quantity we're maximizing to make the fit), and the AIC and BIC statistics. The fixed effects are profiled out of the log-likelihood, so that the log-likelihood is a function only of the data and two parameters: σ_h^2 and σ^2.

2. The formula reminds us of what we asked for: that the forest be a random effect, and that a unique intercept be fit for each level of Forest. The square roots of the estimates of the two parameters are also here.

3. Another metric of model quality is RMSE, which is the estimate of the standard deviation of the response residuals conditional on only the fixed effects. Note that 3.94 is *not* theRMSE, it is instead an estimate of the standard deviation of the response residuals conditional on the fixed and the random effects. Obtaining the RMSE is relatively easy because the random effects and the residuals are assumed to be independent.

$$\text{RMSE} = \sqrt{\sigma_h^2 + \sigma^2} = 4.1$$

The last metric of model quality we can get here is the intra-class correlation. This is the variance of the random effect divided by the sum of the variances of the random effects and the residuals.

$$\rho = \frac{\sigma_h^2}{\sigma_h^2 + \sigma^2} = 0.0788$$

so about 7.9 % of the variation in height (that isn't explained by diameter) is explained by forest. Not very much.

4. Now we have a reminder of the fixed effects model and the estimates of the fixed effects. We have several columns:

   (a) the value of the estimate,

   (b) its standard error (not identical here because of the lack of balance),

   (c) the degrees of freedom (simply mysterious for various reasons),

   (d) the t-value associated with the significance test of the null hypothesis that the estimate is 0 against the two-tailed alternative that it is not 0, which is really rather meaningless for this model, and

   (e) the p-value associated with that rather meaningless test.

5. This is the correlation matrix for the estimates of the fixed effects. It is estimated from the design matrix. This comes from the covariance matrix of the fixed effects, which can be estimated by

$$(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}$$

6. Information about the within-group residuals. Are they symmetric? Are there egregious outliers? Compare these values to what we know of the standard normal distribution, for which the median should be about 0, the first quartile at -0.674, and the third at 0.674.

7. And finally, confirmation that we have the correct number of observations and groups. This is a useful conclusion to draw; it comforts us that we fit the model that we thought we had!

A compact summary of the explanatory power of the model can be had from:

```
> anova(hd.lme.1)

            numDF denDF   F-value p-value
(Intercept)     1    55 2848.4436  <.0001
dbhib.cm        1    55  289.2742  <.0001
```

Deeper design

Let's now treat the Grand fir height/diameter data from Stage (1963) in a different way. We actually have numerous measurements of height and diameter for each tree. It seems wasteful to only use the largest.

Let's still assume that the national forests represent different, purposively selected sources of climatic variation, and that habitat type represents a randomly selected treatment of environment (no, it's probably not true, but let's assume that it is). This is a randomized block design, where the blocks and the treatment effects are crossed. This time we're interested in using all the data. Previously we took only the first measurement. How will the model change? As always, we begin by setting up the data.

```
> require(nlme)
> stage <- groupedData(height.m ~ dbhib.cm | Forest.ID/Tree.ID,
+     data = stage)
```

Let's say that, based on the above information, national forest will be a random effect, and habitat type a candidate fixed effect. So, we'll have anywhere from one to three fixed effects (dbhib, age, and habitat) and two random effects (forest and tree within forest). The response variable will now be the height measurement, nested within the tree, possibly nested within habitat type. Let's assume, for the moment, that the measurements are independent within the tree (definitely not true). Now, let's look to our model. A simple reasonable candidate model is:

$$y_{ijk} = \beta_0 + b_{0i} + b_{0ij} + \beta_1 \times x_{ijk} + \epsilon_{ijk} \tag{2.14}$$

y_{ijk} is the height of tree j in forest i at measurement k, x_{ijk} is the diameter of the same tree. β_0 and β_1 are fixed but unknown parameters, b_{0i} are the forest-specific random and unknown intercepts, and b_{0ij} are the tree-specific random and unknown intercepts. Later we might see if the slope also varies with forest. So, in matrix form, we have:

$$Y = \beta\mathbf{X} + b\mathbf{Z} + \epsilon \tag{2.15}$$

- Y is the vector of height measurements. The basic unit of Y will be a measurement within a tree within a forest. It has 542 observations.
- X will be a matrix of 0s, 1s, and diameters, to allocate the observations to different national forests and different tree diameters at the time of measurement.
- β will be a vector of parameter estimates.
- Z will be a matrix of 0s and 1s to allocate the observations to different forests, and trees within forests.
- b will be a vector of means for the forests and the trees.

- **D** will be a block diagonal matrix comprising a 9×9 identity matrix multiplied by a constant σ_f^2, and then a square matrix for each forest, which will be a diagonal matrix with variances on the diagonals.
- **R** will now be a 542×542 identity matrix multiplied by a constant σ^2.

```
> hd.lme.3 <- lme(height.m ~ dbhib.cm,
+                 random = ~1 | Forest.ID/Tree.ID,
+                 data = stage)
```

Now, the key assumptions that we're making are that:

1. the model structure is correctly specified
2. the tree and forest random effects are normally distributed,
3. the tree random effects are homoscedastic within the forest random effects.
4. the inner-most residuals are normally distributed,
5. the inner-most residuals are homoscedastic within and across the tree random effects.
6. the innermost residuals are independent within the groups.

We again construct diagnostic graphs to check these assumptions. To examine each of the assumptions in turn, I have constructed the earlier suite of graphics, along with some supplementary graphs.

1. an extra qq-plot of the tree-level random effects, to check whether they are normally distributed with constant variance.
   (a) Do the points follow a straight line, or do they exhibit skew or kurtosis?
   (b) Are any outliers evident?
2. a notched boxplot of the tree-level random effects by the grouping variable, to see what the within-group distribution looks like.
   (a) Do the notches intersect 0?
   (b) Is there a trend between the medians of the within-group residuals and the estimated random effect?
3. a scatterplot of the variance of the tree-level random effects within the forest against the forest random effect.
   (a) Is there a distinct positive or negative trend?
4. an autocorrelation plot of the within-tree errors.

As a rule of thumb, we need four plots plus three for each random effect. Cross-reference these against Figures 2.11, 2.12, and 2.13. Each graphic should ideally be examined separately in its own frame. Here's the code:

```
> opar <- par(mfrow = c(1, 3), mar = c(4, 4, 3, 1), las = 1,
+         cex.axis = 0.9)
>   plot(fitted(hd.lme.3, level=0), stage$height.m,
+       xlab = "Fitted Values", ylab = "Observed Values",
+       main = "Model Structure (I)")
>   abline(0, 1, col = "gray")
>   scatter.smooth(fitted(hd.lme.3), residuals(hd.lme.3, type="pearson"),
+       main = "Model Structure (II)",
+       xlab = "Fitted Values", ylab = "Innermost Residuals")
>   abline(h = 0, col = "gray")
> acf.resid <- ACF(hd.lme.3, resType = "normal")
> plot(acf.resid$lag[acf.resid$lag < 10.5],
+      acf.resid$ACF[acf.resid$lag < 10.5],
+      type="b", main="Autocorrelation",
+      xlab="Lag", ylab="Correlation")
> stdv <- qnorm(1 - 0.01/2)/sqrt(attr(acf.resid, "n.used"))
> lines(acf.resid$lag[acf.resid$lag < 10.5],
+       stdv[acf.resid$lag < 10.5],
+       col="darkgray")
> lines(acf.resid$lag[acf.resid$lag < 10.5],
+       -stdv[acf.resid$lag < 10.5],
+       col="darkgray")
> abline(0,0,col="gray")
> par(opar)
```

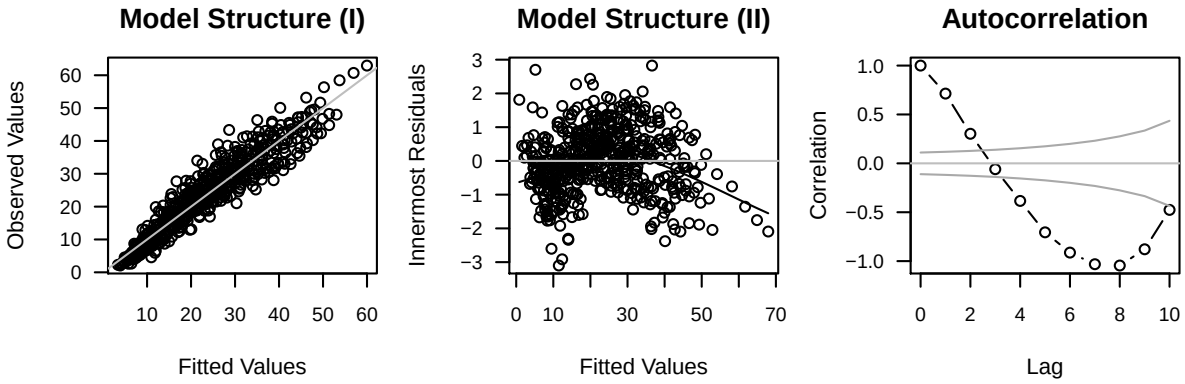

Figure 2.11: Selected overall diagnostics for the mixed-effects fit of the Height and Diameter model for Stage's data.

```
> opar <- par(mfrow = c(1, 3), mar = c(4, 4, 3, 1),
+     las = 1, cex.axis = 0.9)
> ref.forest <- ranef(hd.lme.3, level = 1, standard = T)[[1]]
> ref.tree <- ranef(hd.lme.3, level = 2, standard = T)[[1]]
> ref.tree.frame <- ranef(hd.lme.3, level = 2, augFrame = T,
+     standard = T)
> ref.var.tree <- tapply(residuals(hd.lme.3, type = "pearson",
+     level = 1), stage$Tree.ID, var)
```

```
> ref.var.forest <- tapply(ref.tree, ref.tree.frame$Forest,
+     var)
> qqnorm(ref.forest, main = "QQ plot: Forest")
> qqline(ref.forest)
> qqnorm(ref.tree, main = "QQ plot: Tree")
> qqline(ref.tree)
> qqnorm(residuals(hd.lme.3, type = "pearson"), main = "QQ plot: Residuals")
> qqline(residuals(hd.lme.3, type = "pearson"), col = "red")
> par(opar)
```

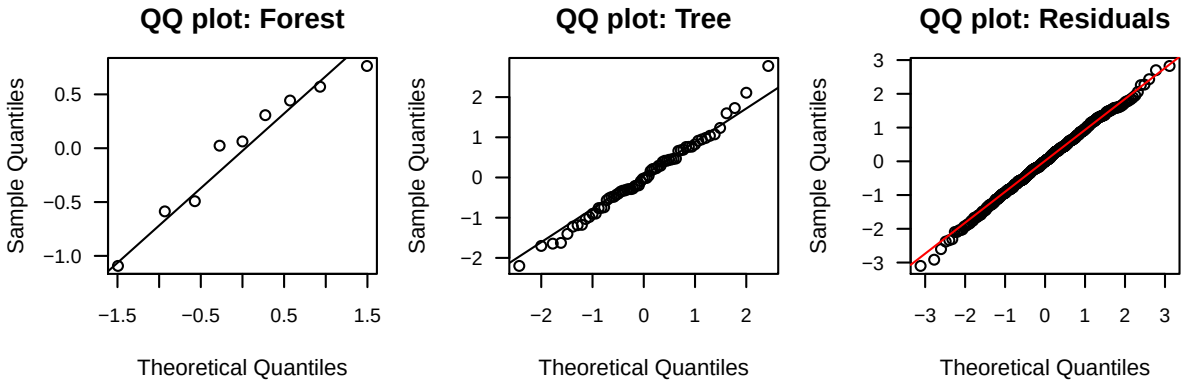

Figure 2.12: Selected quantile-based diagnostics for the mixed-effects fit of the Height and Diameter model for Stage's data.

```
> opar <- par(mfrow = c(2, 2), mar = c(4, 4, 3, 1), las = 1,
+         cex.axis = 0.9)
> boxplot(ref.tree ~ ref.tree.frame$Forest,
+         ylab = "Tree Effects", xlab = "National Forest",
+         notch=T, varwidth = T, at=rank(ref.forest))
> axis(3, labels=format(ref.forest, dig=2), cex.axis=0.8,
+     at=rank(ref.forest))
> abline(h=0, col="darkgreen")
> boxplot(residuals(hd.lme.3, type="pearson", level=1) ~ stage$Tree.ID,
+         ylab = "Innermost Residuals", xlab = "Tree",
+         notch=T, varwidth = T, at=rank(ref.tree))
> axis(3, labels=format(ref.tree, dig=2), cex.axis=0.8,
+     at=rank(ref.tree))
> abline(h=0, col="darkgreen")
> plot(ref.forest, ref.var.forest, xlab="Forest Random Effect",
+     ylab="Variance of within-Forest Residuals")
> abline(lm(ref.var.forest ~ ref.forest), col="purple")
> plot(ref.tree, ref.var.tree, xlab="Tree Random Effect",
+     ylab="Variance of within-Tree Residuals")
> abline(lm(ref.var.forest ~ ref.forest), col="purple")
> par(opar)
```

Everything in these figures look good except for the residual plots and the correlation of the within-tree residuals, which show an unacceptably strong signal. At this point one

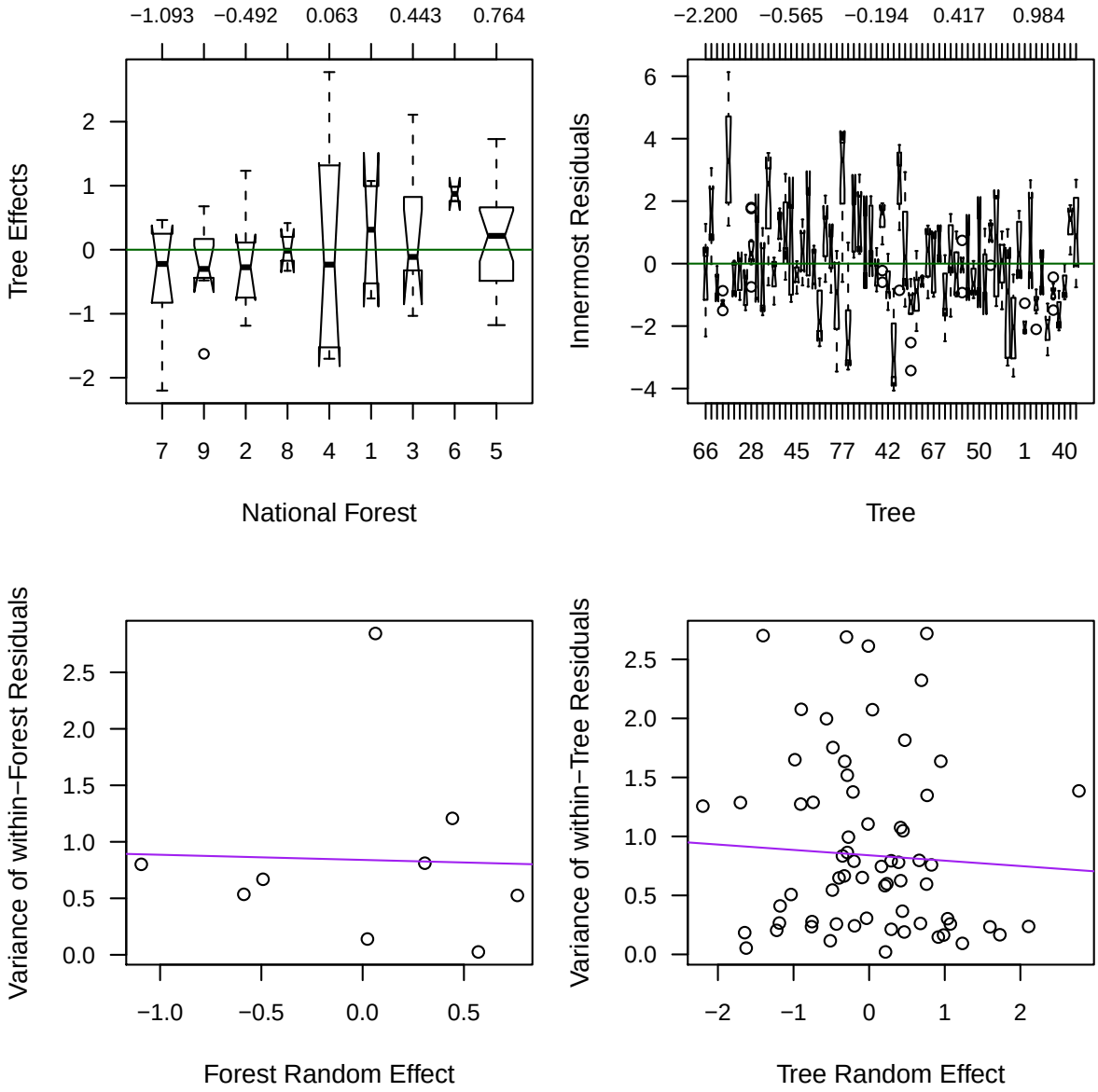

Figure 2.13: Selected random-effects based diagnostics for the mixed-effects fit of the Height and Diameter model for Stage's data.

might think that the next step is to try to fit an autocorrelation function to the within-tree residuals, but the kink in the residual plot suggests that it seems more valuable to take a look at a different diagnostic first.

The augmented prediction plot overlays the fitted model with the observed data, at an optional level within the model. It is constructed using `xyplot()` from lattice graphics, and accepts arguments that are relevant to that function, for further customization. This allows us to sort the trees by national forest, to help us pick up any cluster effects.

```
> trees.in.forests <- aggregate(x = list(measures = stage$height.m),
+     by = list(tree = stage$Tree.ID, forest = stage$Forest.ID),
+     FUN = length)
> panel.order <- rank(as.numeric(as.character(trees.in.forests$tree)))

> plot(augPred(hd.lme.3), index.cond = list(panel.order))
```

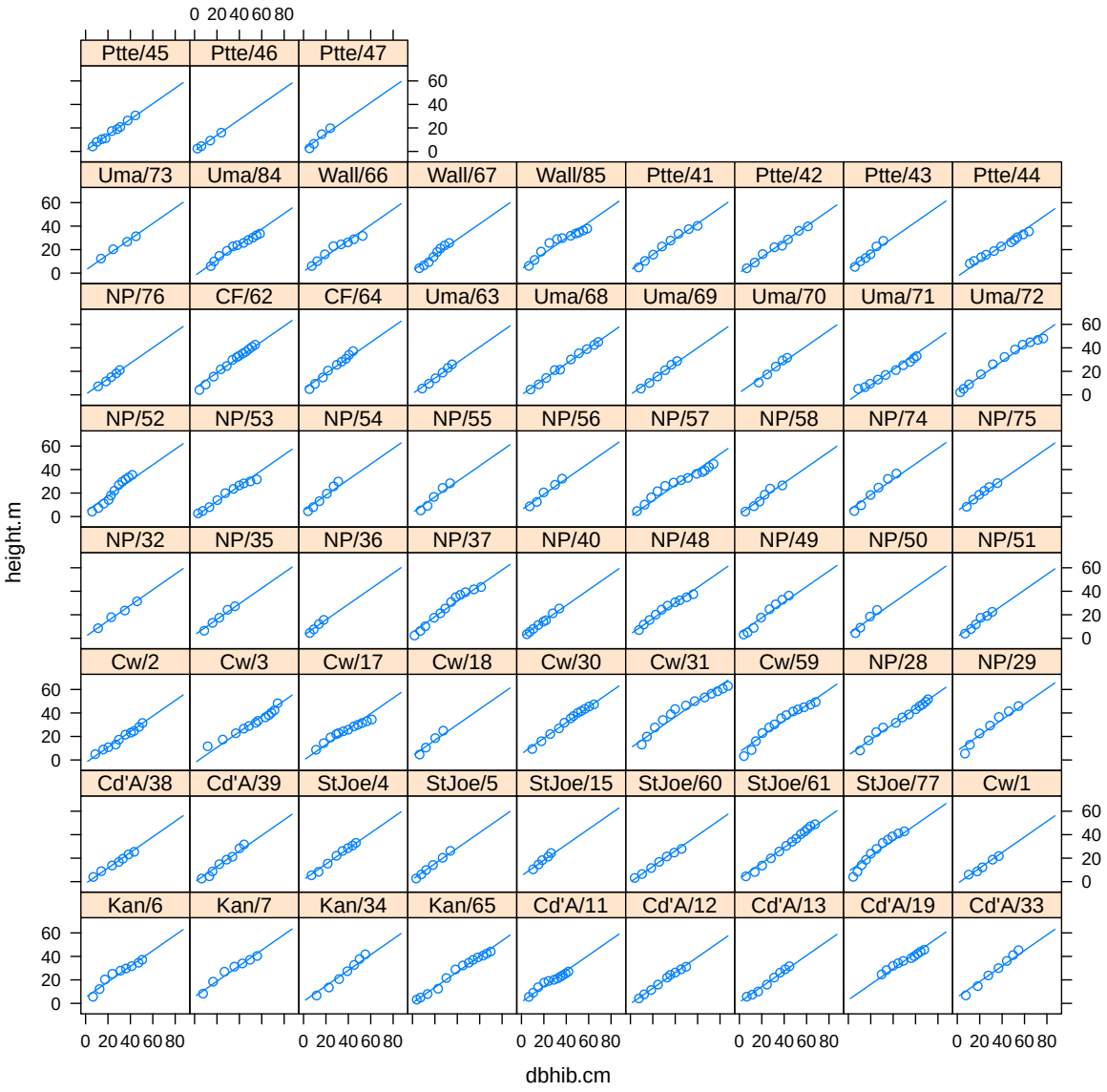

Figure 2.14: Height against diameter by tree, augmented with predicted lines.

The augmented prediction plot (Figure 2.14) shows that a number of the trees have curvature in the relationship between height and diameter that the model fails to pick up, whilst others seem pretty linear. It also shows that the omission of a random slope appears to be problematic.

At this point we have several options, each of which potentially leads to different resolutions for our problem, or, more likely, to several further approaches, and so on. How we proceed depends on our goal. We can:

1. add a quadratic fixed effect;
2. add a quadratic random effect;
3. add quadratic fixed and random effects;
4. correct the model by including a within-tree correlation; and

5. switch to non-linear mixed-effects models and use a more appropriate functional form.

Since we do not believe that the true relationship between height and diameter could reasonably be a straight line, let's add a fixed and a random quadratic diameter effect, by tree, and see how things go. For a start this will increase the number of diagnostic graphs that we want to look at to about 22! We'll show only a sample here.

```
> hd.lme.4 <- lme(height.m ~ dbhib.cm + I(dbhib.cm^2),
+    random = ~ dbhib.cm + I(dbhib.cm^2) | Tree.ID,
+      data = stage)

> opar <- par(mfrow = c(1, 3), mar = c(4, 4, 3, 1), las = 1,
+         cex.axis = 0.9)
>   plot(fitted(hd.lme.4, level=0), stage$height.m,
+       xlab = "Fitted Values", ylab = "Observed Values",
+       main = "Model Structure (I)")
>   abline(0, 1, col = "gray")
>   scatter.smooth(fitted(hd.lme.4), residuals(hd.lme.4, type="pearson"),
+       main = "Model Structure (II)",
+       xlab = "Fitted Values", ylab = "Innermost Residuals")
>   abline(0, 0, col = "gray")
> acf.resid <- ACF(hd.lme.4, resType = "n")
> plot(acf.resid$lag[acf.resid$lag < 10.5],
+      acf.resid$ACF[acf.resid$lag < 10.5],
+      type="b", main="Autocorrelation",
+      xlab="Lag", ylab="Correlation")
> stdv <- qnorm(1 - 0.01/2)/sqrt(attr(acf.resid, "n.used"))
> lines(acf.resid$lag[acf.resid$lag < 10.5],
+       stdv[acf.resid$lag < 10.5],
+       col="darkgray")
> lines(acf.resid$lag[acf.resid$lag < 10.5],
+       -stdv[acf.resid$lag < 10.5],
+       col="darkgray")
> abline(0,0,col="gray")
> par(opar)
```

This has improved the model somewhat, but it looks like we do need to include some accounting for the within-tree correlation. Pinheiro and Bates (2000) detail the options that are available. Also, we'll use `update()` because that starts the model fitting at the most recently converged estimates, which speeds up fitting considerably. Finally, we need to use a different fitting engine, for greater stability.

```
> hd.lme.5 <- update(hd.lme.4, correlation = corCAR1(),
+                    control = lmeControl(opt="optim"))

> opar <- par(mfrow = c(1, 3), mar = c(4, 4, 3, 1),
+     las = 1, cex.axis = 0.9)
> plot(fitted(hd.lme.5, level = 0), stage$height.m,
```

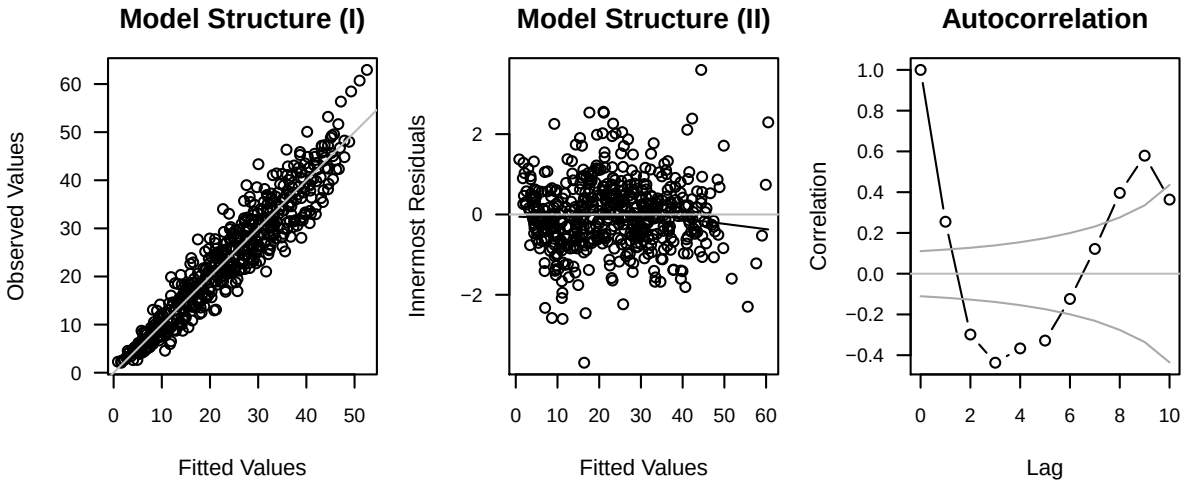

Figure 2.15: Selected diagnostics for the mixed-effects fit of the Height and Diameter model for Stage's data.

```
+     xlab = "Fitted Values", ylab = "Observed Values",
+     main = "Model Structure (I)")
> abline(0, 1, col = "gray")
> scatter.smooth(fitted(hd.lme.5), residuals(hd.lme.5,
+     type = "pearson"), main = "Model Structure (II)",
+     xlab = "Fitted Values", ylab = "Innermost Residuals")
> abline(0, 0, col = "gray")
> acf.resid <- ACF(hd.lme.5, resType = "n")
> plot(acf.resid$lag[acf.resid$lag < 10.5], acf.resid$ACF[acf.resid$lag <
+     10.5], type = "b", main = "Autocorrelation",
+     xlab = "Lag", ylab = "Correlation")
> stdv <- qnorm(1 - 0.01/2)/sqrt(attr(acf.resid, "n.used"))
> lines(acf.resid$lag[acf.resid$lag < 10.5], stdv[acf.resid$lag <
+     10.5], col = "darkgray")
> lines(acf.resid$lag[acf.resid$lag < 10.5], -stdv[acf.resid$lag <
+     10.5], col = "darkgray")
> abline(0, 0, col = "gray")
> par(opar)
```

The correlation is small now.

Another element of the model that we have control over is the variance of the random effects. We haven't seen any red flags for heteroskedasticity in the model diagnostics, so we haven't worried about it. However, such situations are common enough to make an example worthwhile.

Two kinds of heteroskedasticity are common and worthy of concern: firstly, that the variance of the response variable is related to the response variable, and secondly, that the conditional variance of the observations varied within one or more stratum. Some combination of the two conditions is also possible.

We can detect these conditions by conditional residual scatterplots of the following kind. The first is a scatterplot of the innermost Pearson residuals against the fitted values stratified by habitat type. The code to create this graphic is part of the `nlme` package.

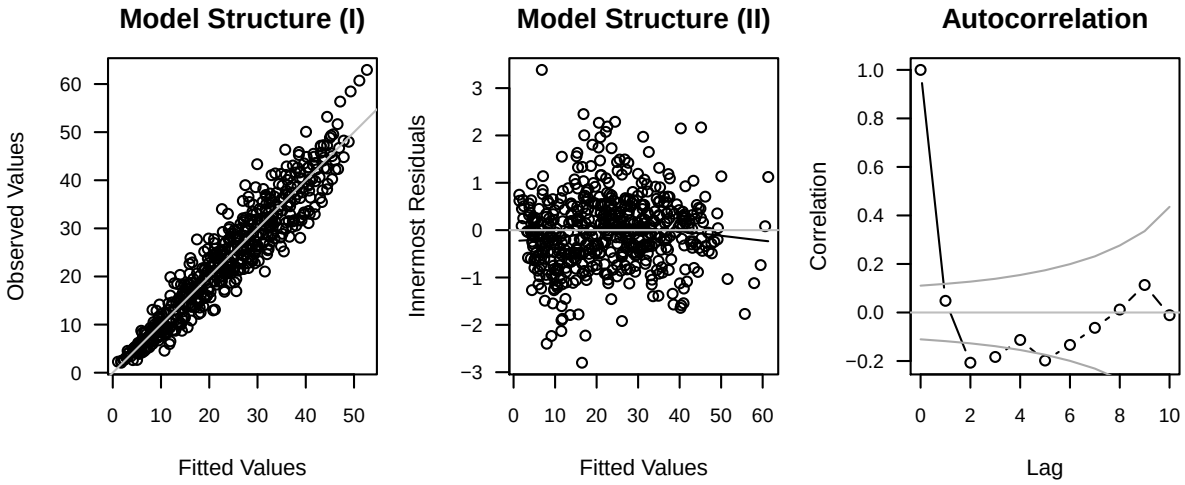

Figure 2.16: Selected diagnostics for the mixed-effects fit of the Height and Diameter model for Stage's data.

```
> plot(hd.lme.5, resid(.) ~ fitted(.) | HabType.ID,
+     layout = c(1, 5))
```

The second is a quantile plot of the innermost Pearson residuals against the normal distribution, stratified by habitat type. This code is provided by the `lattice` package, and we found a template under `?qqmath`.

```
> qqmath(~ resid(hd.lme.5) | stage$HabType.ID,
+        prepanel = prepanel.qqmathline,
+        panel = function(x, ...) {
+          panel.qqmathline(x, distribution = qnorm)
+          panel.qqmath(x, ...)
+        })
```

There seems little evidence in either of figures 2.17 and 2.18 to suggest that the variance model is inadequate.

Had the variance model seemed inadequate, we could have used the `weights` argument in a call to `update` with one of the following approaches:

- `weights = varIdent(form = 1 | HabType.ID)` This option would allow the observations within each habitat type to have their own variance.
- `weights = varPower()` This option would fit a power function for the relationship between the variance and the predicted mean, and estimate the exponent.
- `weights = varPower(form = dbhib.cm | HabType.ID)` This option would fit a power function for the relationship between the variance and the diameter uniquely within each habitat type, and estimate the exponent.
- `weights = varConstPower()` This option would fit a power function with a constant for the relationship between the variance and the predicted mean, and estimate the exponent and constant.

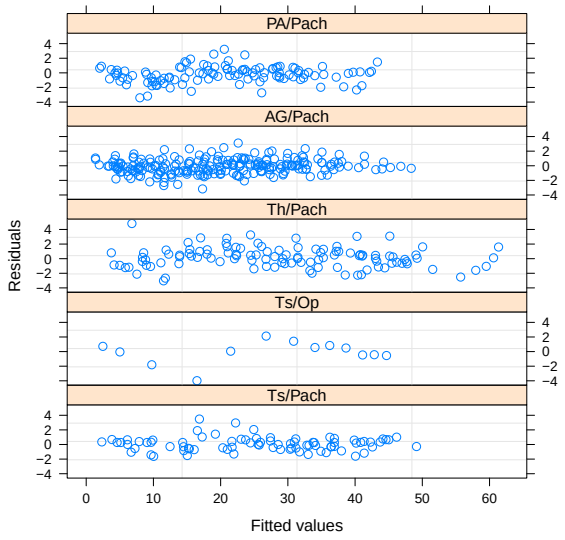

Figure 2.17: Innermost Pearson residuals against fitted values by habitat type.

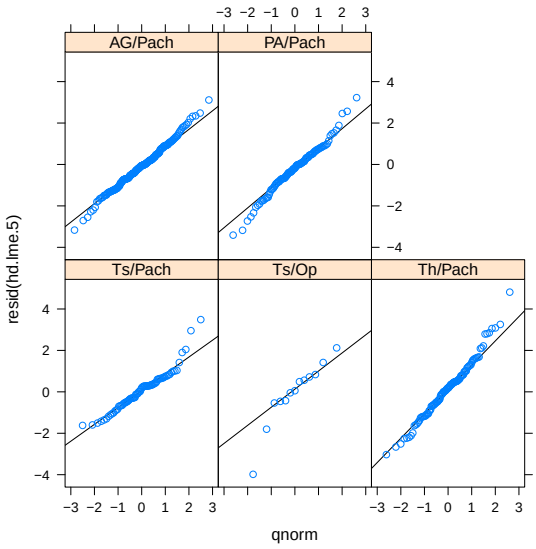

Figure 2.18: Quantile plots of innermost Pearson residuals against the normal distribution, by habitat type.

Other options are available; the function is fully documented in Pinheiro and Bates (2000).

Then, let's accept the model as it stands for the moment. This is the baseline model, as it provides predictions of height from diameter, and satisfies the regression assumptions. Other options may later prove to be better fitting, for example it may be that including habitat type or age in the model obviates our use of the quadratic diameter term. Whether or not this makes for a better model in terms of actual applications will vary!

```
> plot(augPred(hd.lme.5), index.cond = list(panel.order))
```

```
> summary(hd.lme.5)

Linear mixed-effects model fit by REML
 Data: stage
       AIC      BIC    logLik
  1945.521 1992.708 -961.7604

Random effects:
 Formula: ~dbhib.cm + I(dbhib.cm^2) | Tree.ID
 Structure: General positive-definite, Log-Cholesky parametrization
              StdDev       Corr
(Intercept)   0.0007992447 (Intr) dbhb.c
dbhib.cm      0.1844016124 -0.243
I(dbhib.cm^2) 0.0030927949 -0.175 -0.817
Residual      1.4223449956

Correlation Structure: Continuous AR(1)
```

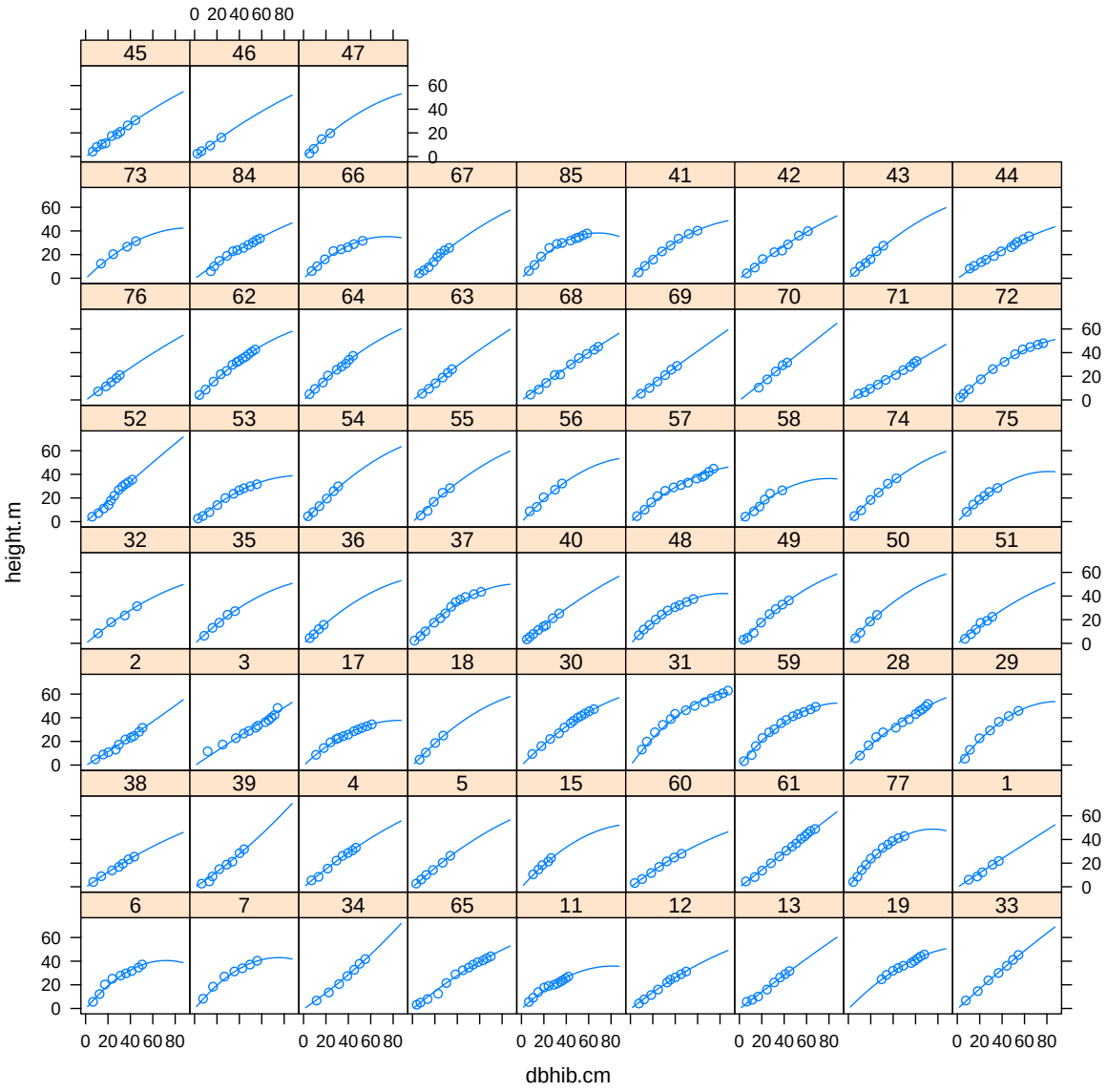

Figure 2.19: Height against diameter by tree, augmented with predicted lines.

```
 Formula: ~1 | Tree.ID
 Parameter estimate(s):
      Phi
0.6660391
Fixed effects: height.m ~ dbhib.cm + I(dbhib.cm^2)
                    Value  Std.Error  DF   t-value p-value
(Intercept)    -0.4959656 0.25444240 474 -1.949226  0.0519
dbhib.cm        0.8918030 0.02985028 474 29.875871  0.0000
I(dbhib.cm^2) -0.0032310 0.00052633 474 -6.138857  0.0000
 Correlation:
              (Intr) dbhb.c
dbhib.cm      -0.514
I(dbhib.cm^2)  0.417 -0.867
```

```
Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-2.79993933 -0.48264758 -0.00876372  0.41456686  3.38442802

Number of Observations: 542
Number of Groups: 66
```

2.4.2 Extensions to the model

We can try to extend the baseline model to improve its performance, based on our knowledge of the system. For example, it might be true that the tree age mediates its diameter - height relationship in a way that has not been captured in the model. We can formally test this assertion, using the `anova` function, or we can examine it graphically, using an added-variable plot, or we can try to fit the model with the term included and see what effect that has on the residual variation.

An added-variable plot is a graphical summary of the amount of variation that is uniquely explained by a predictor variable. It can be constructed in R as follows. Here, we need to decide what level of residuals to choose, as there are several. We adopt the outermost residuals.

```
> age.lme.1 <- lme(Age ~ dbhib.cm, random = ~1 | Forest.ID/Tree.ID,
+       data = stage)
> res.Age <- residuals(age.lme.1, level = 0)
> res.HD <- residuals(hd.lme.5, level = 0)
> scatter.smooth(res.Age, res.HD, xlab = "Variation unique to Age",
+      ylab = "Variation in Height after all but Age")
```

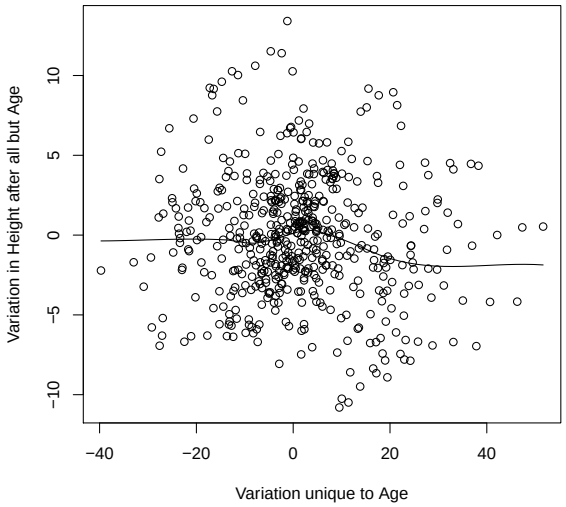

Figure 2.20: Added-variable plot for Age against the ratio of Height over Diameter.

In order to assess whether we would be better served by adding habitat type to the model, we can construct a graphical summary, thus:

```
> xyplot(stage$height.m ~ fitted(hd.lme.5, level=0) | HabType.ID,
+         xlab="Predicted height (m)",
+         ylab="Observed height (m)",
+         data=stage,
+         panel = function(x, y, subscripts) {
+                 panel.xyplot(x, y)
+                 panel.abline(0, 1)
+                 panel.abline(lm(y ~ x), lty=3)
+         }
+ )
```

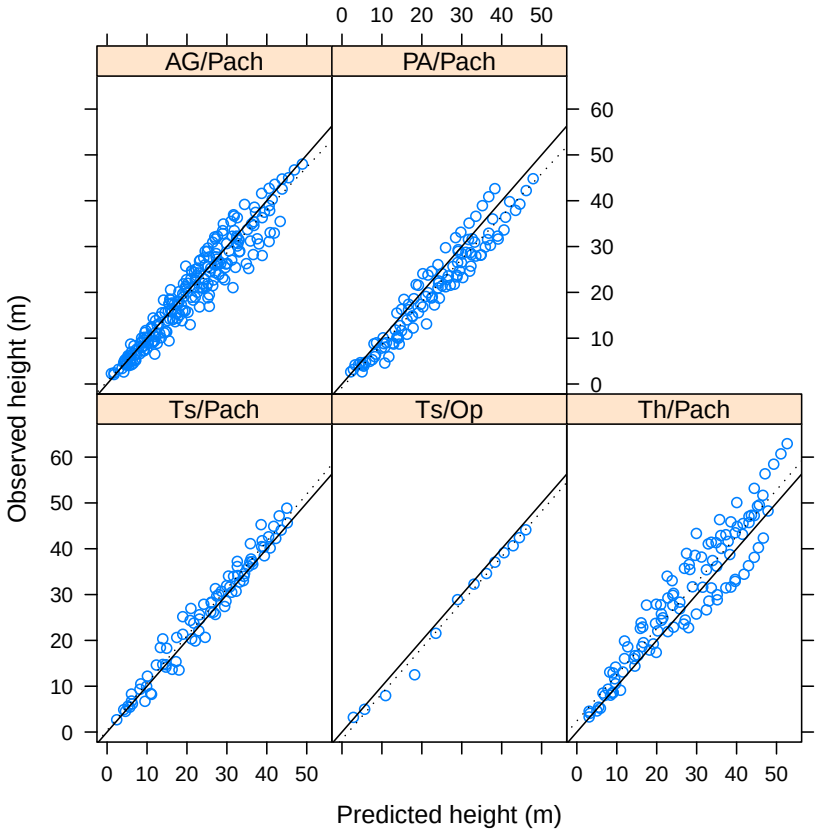

Figure 2.21: Plot of predicted height against observed height, by habitat type. The solid line is 1:1, as predicted by the model. The dotted line is the OLS line of best fit within habitat type.

Neither of figures 2.20 or 2.21 suggest that significant or important improvements would accrue from adding these terms to the model.

The incumbent model represents the best compromise so far. It seems to have addressed most of our major concerns in terms of model assumptions. It may be possible to find a better model with further searching. However, there comes a point of diminishing returns. Note finally that although the presentation of this sequence of steps seems fairly linear, in fact there were numerous blind-alleys followed, much looping, and retracing of steps. This is not a quick process! Introducing random effects to a fixed effects model increases the number of diagnostics to check and possibilities to follow.

2.5 The Model

Let's examine our final model.

$$\begin{aligned} y_{ijk} &= \beta_0 + b_{0i} + b_{0ij} && (2.16)\\ &\quad + (\beta_1 + b_{1i} + b_{1ij}) \times x_{ijk} && (2.17)\\ &\quad + (\beta_2 + b_{2i} + b_{2ij}) \times x_{ijk}^2 && (2.18)\\ &\quad + \epsilon_{ijk} && (2.19) \end{aligned}$$

In matrix form, it is still:

$$\begin{aligned} \mathbf{Y} &= \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b} + \boldsymbol{\epsilon} \\ \mathbf{b} &\sim \mathcal{N}(\mathbf{0}, \mathbf{D}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned}$$

Here's the structure.

- Y is the vector of height measurements. It has 542 observations.
- X is a 3×542 matrix of 1s, diameters and squared diameters.
- β is a vector of length three: it has an intercept, a slope for the linear diameter term, and a slope for the quadratic diameter term.
- Z is a 225×542 unit brute. See below.
- b is a vector of intercepts, and slopes for diameter and diameter squared for each forest, then for each tree. It will be $27 + 198 = 225$ elements long. See below. The predictions can be obtained by `ranef(hd.lme.5)`.
- $\mathbf{D}$ will be a block diagonal matrix comprising 9 3×3 identical matrices, followed by 66 3×3 identical matrices. Each matrix will express the covariance between the 3 random effects within forest or within tree. See below.
- $\mathbf{R}$ will now be a 542×542 symmetric matrix for which the off diagonals are 0 between trees, and a geometric function of the inter-measurement time within trees.

2.5.1 Z

The only role of $\mathbf{Z}$ is to allocate the random effects to the appropriate element. This can be somewhat complicated. Our $\mathbf{Z}$ can be divided into two independent sections; a 27×542 matrix $\mathbf{Z_f}$ associated with the forest level effects, and a 198×542 matrix $\mathbf{Z_t}$ associated with the tree-level effects. In matrix nomenclature:

$$\mathbf{Z} = [\mathbf{Z_f} \mid \mathbf{Z_t}] \quad (2.20)$$

Now, $\mathbf{Z_f}$ allocates the random intercept and two slopes to each observation from each forest. There are 9 forests, so any given row of $\mathbf{Z_f}$ will contain 24 zeros, a 1, and the

corresponding *dbhib* and $dbhib^2$. For example, for the row corresponding to measurement 4 on tree 2 in forest 5, we'll have

$$Z_f = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, d_{524}, d_{524}^2, 0, 0, 0, \ldots) \tag{2.21}$$

Similarly, $\mathbf{Z_t}$ allocates the random intercept and two slopes to each observation from each tree. There are 66 trees, so any given row of $\mathbf{Z_t}$ will contain 195 zeros, a 1, and the corresponding *dbhib* and $dbhib^2$. It will have the same fundamental pattern as above.

2.5.2 b

The purpose of $\mathbf{b}$ is to contain all the predicted random effects. Thus it will be 225 units long, which corresponds to 3 for each level of forest (intercept, slope for diameter, and slope for diameter squared) and 3 for each level of tree (intercept, slope for diameter, and slope for diameter squared).

$$\mathbf{b} = (b_{f10}, b_{f1d}, b_{f1d2}, b_{f20}, b_{f2d}, b_{f2d2}, \ldots, b_{t10}, b_{t1d}, b_{t1d2}, b_{t20}, b_{t2d}, b_{t2d2}, \ldots)' \tag{2.22}$$

The combination of $\mathbf{b}$ and $\mathbf{Z}$ serves to allocate each random effect to the appropriate unit and measurement.

2.5.3 D

Finally, $\mathbf{D}$ dictates the relationships between the different random effects within the levels of forest and tree. We've assumed that the random effects will be independent between habitat types and trees. So, there are only two sub-matrices to this matrix, called $\mathbf{D_f}$ and $\mathbf{D_t}$.

$$\mathbf{D_f} = \begin{bmatrix} \sigma_{bf0}^2 & \sigma_{bf0d} & \sigma_{bf0d2} \\ \sigma_{bf0d} & \sigma_{bfd}^2 & \sigma_{bfdd2} \\ \sigma_{bf0d2} & \sigma_{bfdd2} & \sigma_{bfd2}^2 \end{bmatrix} \tag{2.23}$$

$$\mathbf{D_t} = \begin{bmatrix} \sigma_{bt0}^2 & \sigma_{bt0a} & \sigma_{bt0d2} \\ \sigma_{bt0d} & \sigma_{btd}^2 & \sigma_{btdd2} \\ \sigma_{bt0d2} & \sigma_{btdd2} & \sigma_{btd2}^2 \end{bmatrix} \tag{2.24}$$

Then the structure of $\mathbf{D}$ is simply 9 repetitions of $\mathbf{D_f}$, laid on a diagonal line, followed by 66 repetitions of $\mathbf{D_t}$ laid on the same diagonal, and zeros everywhere else.

Exercise 6 The next steps.

1. Use the tools that we have deployed in this case study to construct a growth model for tree diameter. Carefully document your code, and make sure that you write down a model before you fit it. Does it makes sense? Will you be able to interpret it? Be sure that you check all the necessary diagnostics. And, keep in mind some excellent advice from Schabenberger and Pierce (2002): "don't be afraid to start, and don't be afraid to finish!"
2. Use the tools that we have deployed in this case study to construct a growth model for tree height.

2.6 Wrangling

We observed earlier that the use of the `control` argument was a key tool for the modeller. This element can introduce a little culture shock. Having ourselves come from traditions of model fitting for which exact solutions were easily obtained,and convergence was unequivocal, it was surprising, not to say disheartening, to find that algorithms sometimes quit before convergence. Probably we display our naivete.

The statistical tools that we have been discussing in this chapter are too complicated to admit exact solutions. Accordingly, we have to try to maximize the likelihood, for example, by iterative means. It is necessary and correct that the authors of the code we use will have put in checks, to halt the code in situations where they deem continuing unprofitable.

In any case, bitter experience, and ruthless experimentation have taught us that the code authors do not necessarily have exactly our problem in mind when they are choosing the default parameters for their software. In such cases, it is necessary to roll up our sleeves and plunge our arms into the organs of our analysis. Most of the fitting tools that we use have control arguments that will report or modify the process of model fitting. Experimenting with these will often lead to model configurations that fit reliably.

In short, don't be reluctant to experiment. Any or all of the following strategies might be necessary to achieve a satisfactory fit of your model to your data.

2.6.1 Monitor

In order to be better informed about the progress of model fitting, we use the `msVerbose` argument. It provides a brief, updating description of the progress of the model fit. It will also point out problems along the way, which help the user decide what is the best thing to do next.

2.6.2 Meddle

This strategy involves adjusting the fitting tool.

If the model is failing to converge, then often all that is required is an increase in the number of allowable iterations. The mixed-effects model fitting algorithms in `lme` use a hybrid optimization scheme that starts with the EM algorithm and then changes to Newton-Raphson (Pinheiro and Bates, 2000, p. 80). The latter algorithm is implemented with two loops, so we have three iteration caps. We have found that increasing both `maxIter` and `msMaxIter` is a useful strategy. If we are feeling patient, we will increase them to about 10000, and monitor the process to see if the algorithm still wishes to search. We have occasionally seen iteration counts in excess of 8000 for models that subsequently converged.

We have also had success with changing the optimization algorithm. That is, models that have failed to converge with `nlminb`, by getting caught in a siNgular convergence, have converged successfully using `Nelder-Mead` in `optim`. The default is to use `nlminb`, but it may be worth switching to `optim`, and within `optim`, choosing between `Nelder-Mead`, `BFGS`, and `L-BFGS-B`. Each of these algorithms has different properties, and different strengths and weaknesses. Any might lead more reliably to a satisfactory solution.

2.6.3 Modify

This strategy involves changing the relationship between the model and the data.

The `update` function fits a new model using the output of an old model as a starting point. This is an easy way to set the starting points for parameter estimates, and should be in common use for iterative model building in any case, due to its efficiency. Try dropping out components of the model that complicate it, fitting a smaller, simpler model, and then using `update` to fit the full model.

Alternatively, a number of the model components permit the specification of a starting point. For example, if we provide the `corAR1` function with a suitable number then the algorithm will use that number as a starting point. Specifying this value can help the algorithm converge speedily, and sometimes, at all. Experimenting with subsets of the full model to try to find suitable starting points can be profitable, for example if one has a correlation model and a variance model.

We can also think about how the elements in the data might be interacting with the model. Is the dataset unbalanced, or are there outliers, or is it too small? Any of these conditions can cause problems for fitting algorithms. Examining the data before fitting any model is standard practice. Be prepared to *temporarily* delete data points, or augment under-represented portions, in order to provide the `update` function with a reasonable set of starting values.

2.6.4 Compromise

Sometimes a model involves a complicated hierarchy of random effects. It is worth asking whether or not such depth is warranted, and whether a superficially more complex, but simpler model, might suffice. The case study in this chapter serves as a good example: although model fit benefited by allowing each individual tree to have a random slope, there was no need to allow each national forest to have a random slope. Including a slope for each forest made the model unnecessarily complicated, and also made fitting the model much harder. Specifying the smaller model was a little less elegant, however.

Finally, sometimes no matter what exigencies we try, a model will not converge. There is a point in every analysis where we must decide to cut our losses and go with the model we have. If we know that the model has shortcomings, then it is our responsibility to draw attention to those shortcomings. For example, if we are convinced that there is serial autocorrelation in our residuals, but cannot achieve a reasonable fit using the available resources, then providing a diagnostic plot of that autocorrelation is essential. Furthermore, it is important to comment on the likely effect of the model shortcoming upon inference and prediction. If we are fortunate enough to be able to fit a simpler model that does include autocorrelation, for example, we might demonstrate what effect the inclusion of that portion of the model has upon our conclusions. We would do this by fitting three models: the complex model, the simple model with the autocorrelation, and the simple model without the autocorrelation. If the difference between the latter two models is modest, then we have some modest degree of indirect evidence that perhaps our conclusions will be robust to misspecification of the complex model. It is not ideal, but we must be pragmatic.

2.7 Appendix - Leave-One-Out Diagnostics

Another important question is whether there are any outliers or high-influence points. In a case like this it is relatively easy to see from the diagnostics that no point is likely to dominate the fit in this way. However, a more formal examination of the question is valuable. To date, there is little peer-reviewed development of the problem of outlier and influence detection. Schabenberger (2005) provides an overview of the extensive offerings available in SAS, none of which are presently available in R. Demidenko and Stukel (2005) also provide some alternatives.

The simplest thing, in the case where a model fit is relatively quick, is to refit the model dropping each observation one by one, and collecting the results in a vector for further analysis. This is best handled by using the `update()` function.

```
> all.betas <- data.frame(labels=names(unlist(hd.lme.1$coefficients)))
> cook.0 <- cook.1 <- rep(NA, dim(stage.old)[1])
> p.sigma.0 <- length(hd.lme.1$coefficients$fixed) *
+          var(residuals(hd.lme.1, level=0))
> p.sigma.1 <- length(hd.lme.1$coefficients$fixed) *
+          var(residuals(hd.lme.1, level=1))
> for (i in 1:dim(stage.old)[1]) {
+   try({ hd.lme.n <- update(hd.lme.1, data = stage.old[-i,])
+         new.betas <- data.frame(labels=names(unlist(hd.lme.n$coefficients)),
+                                 coef=unlist(hd.lme.n$coefficients))
+         names(new.betas)[2] <- paste("obs", i, sep=".")
+         all.betas <- merge(all.betas, new.betas, all.x = TRUE)
+         cook.0[i] <- sum((predict(hd.lme.1, level=0, newdata=stage.old) -
+                           predict(hd.lme.n, level=0, newdata=stage.old))^2) /
+                             p.sigma.0
+         cook.1[i] <- sum((predict(hd.lme.1, level=1, newdata=stage.old) -
+                           predict(hd.lme.n, level=1, newdata=stage.old))^2) /
+                             p.sigma.1
+       })
+ }
```

We can then examine these results with graphical diagnostics (Figures 2.22 and 2.23). The Cook's Distances presented here are only approximate.

```
> all.betas <- t(all.betas[, -1])
> len.all <- length(unlist(hd.lme.1$coefficients))
> len.fixed <- length(hd.lme.1$coefficients$fixed)
> len.ran <- length(hd.lme.1$coefficients$random$Forest.ID)

> opar <- par(mfrow=c(len.all, 1), oma=c(2,0,1,0), mar=c(0,4,0,0), las=1)
> for (i in 1:len.fixed) {
+   plot(all.betas[,i], type="l", axes=F, xlab="", ylab="")
+   text(length(all.betas[,i])-1, max(all.betas[,i], na.rm=T),
+        names(unlist(hd.lme.1$coefficients))[i],
+        adj=c(1,1), col="red")
+   axis(2)
```

```
+   box()
+ }
> for (i in (len.fixed+1):(len.all)) {
+   plot(all.betas[,i], type="l", axes=F, xlab="", ylab="")
+   text(length(all.betas[,i])-1, max(all.betas[,i], na.rm=T),
+        names(unlist(hd.lme.1$coefficients))[i],
+        adj=c(1,1), col="red")
+   axis(2)
+   box()
+ }
> axis(1)
> par(opar)
```

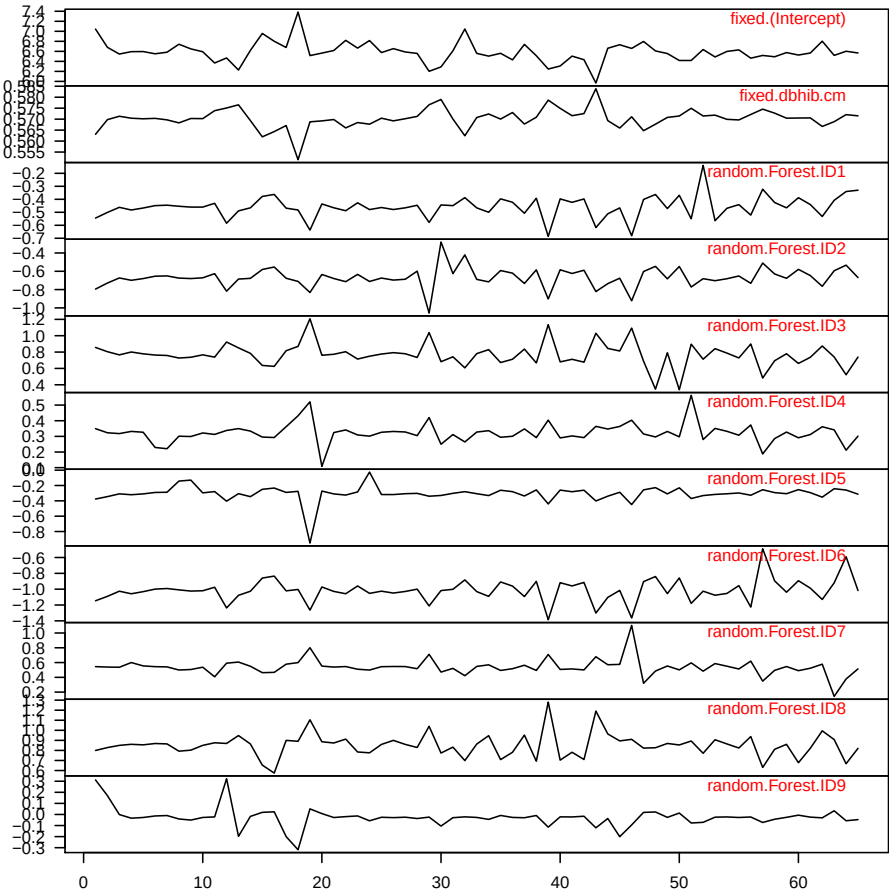

Figure 2.22: The parameter estimates for the fixed effects and predictions for the random effects resulting from omitting one observation.

```
> cook <- data.frame(id=stage.old$Tree.ID, fixed=cook.0, forest=cook.1)
> influential <- apply(cook[,2:3], 1, max) > 1
> plot(cook$fixed, cook$forest, type="n",
+     xlab="Outermost (Fixed effects only)",
+     ylab="Innermost (Fixed effects and random effects)")
> points(cook$fixed[!influential], cook$forest[!influential])
> if(sum(influential) > 0)
```

```
+    text(cook$fixed[influential], cook$forest[influential],
+       cook$id[influential], col="red", cex=0.85)
```

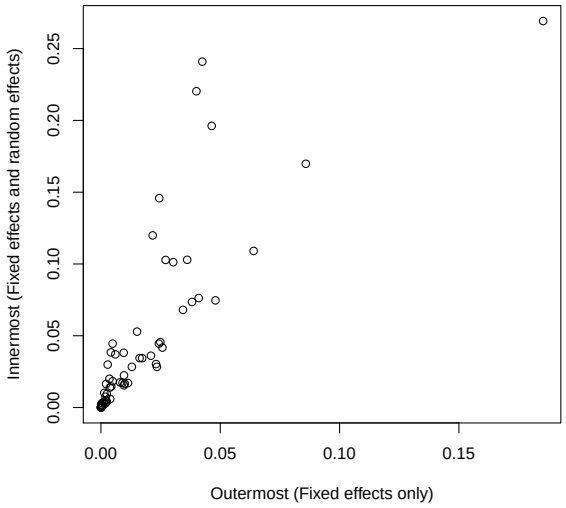

Figure 2.23: Cook's Distances for outermost and innermost residuals. Values greater than 1 appear in red and are identified by the tree number. The corresponding observations bear further examination.

And, what about the removal of entire forests? We can compute the effects similarly.

Chapter 3

GLMM

In previous sessions, we have covered the analysis of two distinct extensions of the linear model: first, when the response was, conditionally, distributed according to a member of the exponential family (generalised linear models), and second, models for which the response was a function of both fixed and random effects (mixed-effects models). In this session, we bring these two topics together, with a moderately light touch, to allow for models that combine these characteristics (generalised linear mixed-effects models, glmm).

Numerous books and articles have been written about glmm, but the science is yet to settle. Bolker et al. (2009) has some nice reading. Here, we wander through a brace of problems that glmm can be used to tackle.

The package that we deploy is lme4 (Bates and Maechler, 2010). This package is under development, and some functionality is missing for some models — sadly, the glmm offerings are still relatively nascent.

```
> library(lme4)
```

3.1 Wabbits

Dana Sanchez, one-time PhD student in wildlife at the University of Idaho[1], did her dissertation on various aspects of the ecology of pygmy rabbits (*Brachylagus idahoensis*). In the analysis picked up for this exercise, Dana was interested in the changes of status of pygmy rabbit burrows as a function of the status of the burrow, the location, the season, and the year. The work was published in Sanchez et al. (2009)[2].

3.1.1 Burrow Entrance Count

Our first goal is to explain variation in the changes in the number of active burrow entrances, with specific attention to the effects of season and location. The data comprise 60 burrows, 30 in each of two locations, which have been visited up to 8 times. It is reasonable to fit a model where the unit of replication is the burrow, rather than a burrow visit by the researcher, that is, the burrow should be a random effect. The tentative model will be:

[1] Now Dr. Sanchez!

[2] I have permission to use the data for the purposes of demonstration, but they are otherwise encumbered.

$$y_{ij} \sim \text{Poisson}(\lambda_{ij}) \tag{3.1}$$

$$\text{where} \log \lambda_{ij} = \alpha_i + \beta_j + b_i + c_j \tag{3.2}$$

where y_{ij} is the count of active burrow entrances for burrow i at measurement j, α_i is a factor with two levels that depends on the location of burrow i (CG or RC), β_j is a factor with four levels that depends on the season of measurement time j, b_i is a burrow-specific random effect to capture burrow-specific variation, and t_j is a year-specific random effect to capture year-specific variation. Note that here we have crossed random effects: burrow and year.

We will also assume that:

- The linear form of the model is appropriate.
- the b_i are independent and identically distributed; $b_i \sim \mathcal{N}(0, \sigma_b^2)$

We brush over the preparatory steps of using lattice or ggplot2 to construct conditioning plots with the data and smooth lines. However, clever use of graphics to guide model choice and communicate the model outcome are very valuable.

Note that the model lacks a residual term, and also any assumption about the distribution of the residuals. This characteristic is a key distinction of the glm: inference is performed on the conditional distribution of the observations. We fit the model as follows.

```
> ent.glmm.1 <- lmer(count ~ season + location + (1|year) + (1|burrow),
+                            family = poisson,
+                            data = burrows)
```

Most of the function call is self-explanatory in the context of mixed-effects or generalized linear models. The model formula contains both the fixed and the random effects; the random effects are denoted as `(1|burrow)`, which asks for each burrow to have a random intercept and `(1|year)`, which asks for each year to have its own random intercept. We are also required to nominate a member of the exponential family for the conditional distribution.

As always, we need to assess the fit of the model before we proceed further. This assessment involves two substantial components: checking the conditional distribution of the response variable, and checking the distribution of the random effects.

In order to check the conditional distribution of the response variable, recall that the distribution is poisson, conditional on the fixed and the random effects. We can obtain the linear predictor, which comprises both the fixed and random effects, by

```
> eta.hat <- ent.glmm.1@eta
```

Note that we used `@` to extract the matrix, instead of the `$` with which we have been more familiar. This is a consequence of the lme4 package supporting S4 classes instead of S3 classes, and is irrelevant but necessary. This quantity is:

$$\hat{\eta}_{ij} = \hat{\alpha}_i + \hat{\beta}_j + \hat{b}_i + \hat{c}_j \tag{3.3}$$

We now transform the linear predictor using the link function:

```
> y.ij.hat <- exp(eta.hat)
```

The model assumption is now that all our observations are conditionally poisson with this variable as the mean. We can assess this assumption a few ways. The most compact and useful is to provide a scatterplot of the predicted values against the observed values, and augment the plot with some lines that reflect increases in the underlying variation. Recall that the mean and the variance for the poisson distribution are the same. Therefore, we would expect to see about half the data appearing within the bands $\hat{y}_{ij} \pm \sqrt{\hat{y}_{ij}}$, and most of the data within the bands $\hat{y}_{ij} \pm 2 \cdot \sqrt{\hat{y}_{ij}}$.

We produce this graph, presented in Figure 3.1, using the following code. The figure suggests that the nominated (lienar) functional form is appropriate, but that there is some under-dispersion; that is, the variation of the observations conditional on the model is too small. Almost all of the data are within one standard deviation of the mean.

```
> par(las = 1)
> plot(y.ij.hat, burrows$count, xlab = "Fitted Count", ylab = "Observed Count")
> abline(0, 1, col = "red")
> curve(x - sqrt(x), col = "red", lty = 2, add = TRUE)
> curve(x + sqrt(x), col = "red", lty = 2, add = TRUE)
> curve(x - 2 * sqrt(x), col = "red", lty = 2, add = TRUE)
> curve(x + 2 * sqrt(x), col = "red", lty = 2, add = TRUE)
```

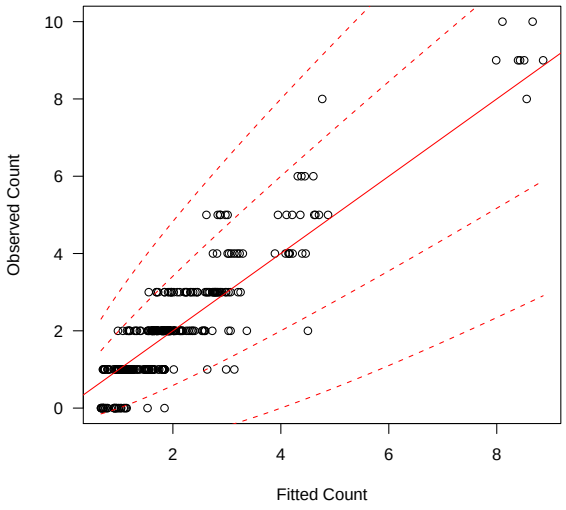

Figure 3.1: Distributional check for conditional poisson model. The fitted count includes the fixed and random effects. The solid red line is $x = y$. The inner dashed lines are one standard deviation from $x = y$; the outer dashed lines are two standard deviations from $x = y$.

We also need to check the assumption of normality of the random effects. This diagnostic graph is presented in Figure 3.2, and is computed using the following code:

```
> qqmath(ranef(ent.glmm.1, post = TRUE))[["burrow"]]
```

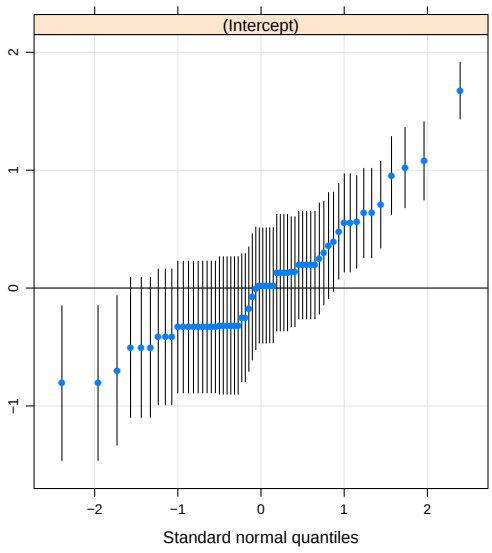

Figure 3.2: Diagnostic plot for the basic model of burrow active entrance count. This plot is a qq-norm plot of the predicted random effects with their prediction intervals.

The assumption of normality seems questionable — there is evidence that the left tail is shorter and the right tail longer, characteristic of skewed distributions.

We're not entirely satisfied by the model, so we'll proceed only tentatively. In a modelling exercise we may try other terms, transformations, etc., but here we will simply pledge to interpret all subsequent statistics in the light of the possible failure of our assumptions to match the data-model interplay. We examine the model by

```
> print(ent.glmm.1, correlation = FALSE)

Generalized linear mixed model fit by the Laplace approximation
Formula: count ~ season + location + (1 | year) + (1 | burrow)
   Data: burrows
   AIC   BIC logLik deviance
 273.1 302.0 -129.5    259.1
Random effects:
 Groups Name        Variance  Std.Dev.
 burrow (Intercept) 0.3048711 0.552151
 year   (Intercept) 0.0027634 0.052568
Number of obs: 462, groups: burrow, 60; year, 3

Fixed effects:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.449274   0.162590   2.763  0.00572 **
seasonspring -0.009036   0.138133  -0.065  0.94785
seasonsummer  0.005413   0.133540   0.040  0.96767
seasonwinter  0.040413   0.137322   0.294  0.76853
locationRC   -0.002058   0.161621  -0.013  0.98984
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This output brings us to the saddest point of the statistician's work. After all that modelling effort, the compromise, the double-checking, and the reqriting of code to make the model make sense to us, we get dumped with tiny estimates, large standard errors, and consequently no action from the Wald normal tests. Still, at least we weren't out in the field. Sorry Dana!

These estimates are of the parameters in the linear predictor, so they have only an indirect interpretation in the context of the count. The coefficients are estimates of the (partial) gradient of the predictor variable against the log of the entrance count.

3.1.2 Burrow Status

We seek to explain variation in the changes in the status of the burrows, with specific attention to the effects of season and location. The data comprise 60 burrows, 30 in each of two locations, which have been visited up to 8 times. It is reasonable to fit a model where the unit of replication is the burrow, rather than a burrow visit. Now, the response variable will be whether or not the status of the burrow changes from one observation to another, and the model will be designed to assess the effects of season and location upon the transition probability. Only two burrow classes are considered. Our observations are now Bernoulli distributed, 1 for a change in burrow status, and 0 for no change. The tentative model will be:

$$E(y_{ij}|b_i) = \pi(x_{ij}) = \frac{\exp(\eta_{ij})}{1 + \exp(\eta_{ij})} \tag{3.4}$$

where

$$\eta_{ij} = \alpha_j + \beta_j + \lambda_i + \alpha_j\beta_j + \alpha_j\lambda_i + c_j + b_i \tag{3.5}$$

and

$$y_{ij}|b_i \sim \text{Independent Bernoulli}[\pi(x_{ij})] \tag{3.6}$$

where y_{ij} is change in burrow status from measurement $j-1$ to measurement j for burrow i, α_j is a factor with two levels that depends on the status of the burrow at time $j-1$, λ_i is a factor with two levels that represents the location of burrow i (CG or RC), β_j is a factor with four levels that depends on the season of the timespan from measurement $j-1$ to measurement j, c_j is a two-level random effect that represents the year, and b_i is a burrow-specific random effect, $b_i \sim \mathcal{N}(0, \sigma_b^2)$, to capture burrow-specific propensity for change. Interactions are indicated by juxtaposition. We will assume that:

- The linear form of the model is appropriate.
- the b_i are independent and identically distributed; $b_i \sim \mathcal{N}(0, \sigma_b^2)$

As before, we skip over the panel graphics that we would ordinarily use to support our choices of functional form. We fit this model by

```
> rob.glmm.1 <- lmer(status.change ~ status.p * (season + location) +
+     (1 | year) + (1 | burrow), family = binomial, data = burrow.changes)
```

Figure 3.3: Distributional check for conditional binomial model. The fitted count includes the fixed and random effects. The solid red line is $x = y$. The inner dashed lines are one standard deviation from $x = y$; the outer dashed lines are two standard deviations from $x = y$.

In order to check the conditional distribution of the response variable, recall that the distribution is binomial, conditional on the fixed and the random effects. Developing graphics to reliably assess the former is a work in progress.

We also need to check the assumption of normality of the random effects. This diagnostic graph is presented in Figure 3.4, and is computed using the following code:

```
> qqmath(ranef(rob.glmm.1, post = TRUE))[["burrow"]]
```

The assumption of normality seems questionable — there is evidence that the left tail is shorter and the right tail longer, characteristic of skewed distributions.

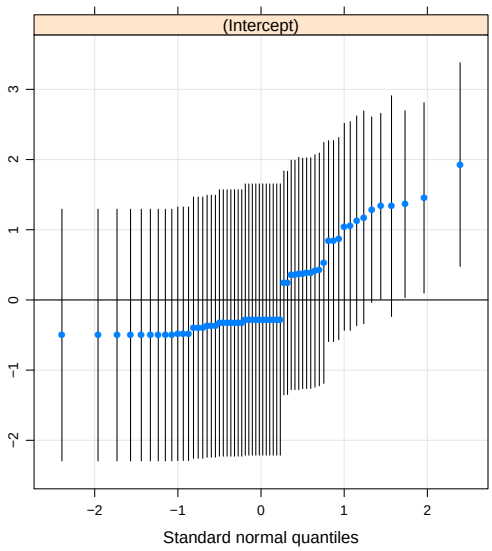

Figure 3.4: Diagnostic plot for the basic model of burrow status. This plot is a qq-norm plot of the predicted random effects with their prediction intervals.

We're not entirely satisfied by the model, so we'll proceed only tentatively. In a modelling exercise we may try other terms, transformations, etc., but here we will simply pledge to interpret all subsequent statistics in the light of the possible failure of our assumptions to match the data-model interplay. We examine the model by

```
> print(rob.glmm.1, correlation = FALSE)
```

```
Generalized linear mixed model fit by the Laplace approximation
Formula: status.change ~ status.p * (season + location) + (1 | year) +      (1 | burr
   Data: burrow.changes
   AIC   BIC logLik deviance
 276.5 324.0 -126.3    252.5
Random effects:
```

```
 Groups Name        Variance Std.Dev.
 burrow (Intercept) 1.2312   1.1096
 year   (Intercept) 0.0000   0.0000
Number of obs: 384, groups: burrow, 60; year, 2

Fixed effects:
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)             -2.330812   0.729421  -3.195  0.00140 **
status.p2               -0.312818   1.136885  -0.275  0.78320
seasonspring            -0.190878   0.822846  -0.232  0.81656
seasonsummer            -0.001818   0.814237  -0.002  0.99822
seasonwinter             0.297448   0.776160   0.383  0.70155
locationRC              -0.433380   0.594122  -0.729  0.46573
status.p2:seasonspring  -0.271483   1.290306  -0.210  0.83335
status.p2:seasonsummer   0.246948   1.244117   0.198  0.84266
status.p2:seasonwinter  -1.441174   1.420841  -1.014  0.31043
status.p2:locationRC     0.996747   0.842522   1.183  0.23679
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sadly, the story is much the same as above.

3.2 Exercise: Cow Flu

Contagious bovine pleuropneumonia (CBPP) is a major disease of cattle in Africa, caused by a mycoplasma. This dataset describes the serological incidence of CBPP in zebu cattle during a follow-up survey implemented in 15 commercial herds located in the Boji district of Ethiopia. The goal of the survey was to study the within-herd spread of CBPP in newly infected herds. Blood samples were quarterly collected from all animals of these herds to determine their CBPP status. These data were used to compute the serological incidence of CBPP (new cases occurring during a given time period). Some data are missing (lost to follow-up).

These data are provided within the `lme4` package.

```
> data(cbpp)
> m1 <- glmer(cbind(incidence, size - incidence) ~ 0 + period +
+     (1 | herd), nAGQ = 5, cbpp, binomial)
```

The mysterious argument is `nAGQ`, which specifies the number of points to use for adaptive Gaussian quadrature. Explaining the use and choice of this argument requires a bit of background. We use maximum likelihood as the statistical fitting technique to determine the parameter estimates for our model. The tricky part is that in order to be able to evaluate, and thus maximize, the likelihood, we need to estimate all the parameters. In the case of mixed-effects models, "all the parameters" includes the random effects. This inclusion is problematic because we chose the random effects to be random precisely because we didn't want to be burdened with their estimation! There is no such dilemma in the normal case (previously covered) because the functional form of the normal distribution allows the evaluation of the likelihood to be (relatively) conveniently handled.

Now, however, there is no convenient solution. lme4 adopts integration: the effect of the random effects upon the likelihood is to be integrated out. This process would be very expensive in computer time, so instead of performing the full integration, we use an approximation called *quadrature* to estimate the integral. Quadrature bascially involves using a weighted function of the value of the function at a set of k points. Obviously, the higher the number of points at which the function is evaluated, the more accurate the integral estimate will be, and the longer the process will take. The `nAGQ` argument tells R how many points to use. If the argument is omitted then R will use the Laplacian approximation to the integral, which is the same as taking `nAGQ = 1`.

From a practical point of view, all you need to know is that approximations are being used to make our lives easier, and it might be worthwhile fiddling with this parameter and seeing if higher values change the inference. At present, this value can only be greater than 1 if there is only one grouping factor. Other than that, we move on ...

Chapter 4

Showcase: equivalence tests

4.1 Introduction

Model validation and environmental monitoring are essential components of the management and protection of biodiversity and natural resources. Yet, validation and monitoring are poorly served by traditional, established statistical tools.

Equivalence tests, described in detail below, are a relatively new branch of statistics. Equivalence tests are ideal for demonstrating that predictions from a model conform to field observations, or that field observations conform to legislated norms. Equivalence tests are more appropriate for these tasks than the traditional hypothesis tests.

A characteristic of the traditional hypothesis tests is that they start from the position that two things are the same, and try to prove otherwise (that is, they are designed to *split*). Equivalence tests start from the position that two things are different, and try to prove that they are the same. That is, equivalence tests are designed to *lump*.

The goal of both validation and monitoring is to compare measurements with some expectation, whether those be from model predictions, as in the case of a model validation exercise, or from a nominal standard, as in the case of environmental or natural resource monitoring. In each case, it is desirable to be able to conclude that the predictions and the observations are the same, or that the field observations meet the required standard.

Of course, we can never conclude that these quantities are identical, instead we wish to conclude that they are sufficiently similar that differences are irrelevant.

Various authors have recommended the use of equivalence tests instead of the traditional hypothesis tests for specific situations, for example in environmental management (McBride, 1999), biology (Parkhurst, 2001), ecology (Dixon and Pechmann, 2005), medical studies (see Berger and Hsu, 1996, among many others), psychology (Rogers et al., 1993), and model validation (Robinson and Froese, 2004).

Some equivalence tests are available in R via the equivalence package.

```
> library(equivalence)
```

4.2 TOST 1

A popular equivalence test is an intersection-union test called the TOST, for Two-One-Sided Tests, which can be used to try to assess the equivalence between the means of two populations based only on a sample from each (Schuirmann, 1981; Westlake, 1981). The TOST is applied as follows.

1. Choose
   (a) a test statistic, for example, the mean of the differences between the samples,
   (b) a *size*, which is essentially the probability of a false positive for the test, e.g. $\alpha = 0.05$, and
   (c) a *region of similarity*, which is a span of numbers within which we think that if the two population means are this far apart then we can treat them as though they were the same.
2. Compute the statistic and two one-sided confidence intervals, one lower and one upper, each with nominal coverage equal to $1 - \alpha$.
3. If the overlap of the two intervals computed in step 2 is *inside* the region determined in step 1c then conclude that the means of the populations are statistically similar.

This test is easy to apply and interpret, although it is not the most powerful available (Berger and Hsu, 1996; Wellek, 2003). Here, we run the TOST in R using the equivalence package.

We have a dataset of height and diameter measurements taken from the University of Idaho Experimental Forest (UIEF). In the summer of 1991, a stand examination was made of the Upper Flat Creek unit of the UIEF. 144 plots were located on a square grid, with north-south inter-plot distance of 134.11 m and east-west inter-plot distance of 167.64 m. A 7.0 m^2/ha BAF variable-radius plot was installed at each plot location. Every tree in the plot was measured for species and diameter at 1.37 m (dbh), recorded in mm, and a subsample of trees was measured for height, recorded in dm. The inventory data are stored in the `ufc` object, which is provided by the equivalence package.

Our goal is to assess whether the Prognosis height-diameter model is suitable for this stand. We will assume that the sampling units are the trees, and we will ignore the clustering in the sample design for the purposes of this demonstration, this point is discussed further below.

```
> data(ufc)
> str(ufc, strict.width = "cut")

'data.frame':        633 obs. of  10 variables:
 $ Plot       : int  1 2 2 3 3 3 3 3 3 ...
 $ Tree       : int  1 1 2 1 2 3 4 5 6 7 ...
 $ Species    : Factor w/ 13 levels "","DF","ES","F",..: 1 2 12 11 6..
 $ Dbh        : int  NA 390 480 150 520 310 280 360 340 260 ...
 $ Height     : int  NA 205 330 NA 300 NA NA 207 NA NA ...
 $ Height.ft  : num  NA 67.3 108.3 NA 98.4 ...
 $ Dbh.in     : num  NA 15.4 18.9 5.9 20.5 ...
 $ Height.ft.p: num  NA 83.5 107.3 44.2 106.1 ...
 $ Height.m   : num  NA 20.5 33 NA 30 NA NA 20.7 NA NA ...
 $ Height.m.p : num  NA 25.4 32.7 13.5 32.3 ...
```

The height predictions have already been computed for the trees. Hence our goal is to perform an equivalence test on the two variables: measured (`Height.m`) and predicted (`Height.m.p`) heights, both expressed in metres. Let's check these data first.

```
> lapply(ufc[, c("Height.m", "Height.m.p")], summary)

$Height.m
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
    0.0    18.7    24.0    24.0    29.0    48.0   248.0

$Height.m.p
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  8.921  20.100  25.240  24.260  28.980  46.350  18.000
```

Our test involves computing a confidence interval for the differences of two means. We'll need to invoke the central limit theorem (CLT) in order to compute the confidence interval. Sometimes we have to do this on blind faith, but here we have a sample of differences that we can assess for proximity to normality. We provide the qq-norm plot in Figure 4.1, using the following code.

```
> error.hats.m <- ufc$Height.m - ufc$Height.m.p
> qqnorm(error.hats.m)
> qqline(error.hats.m)
```

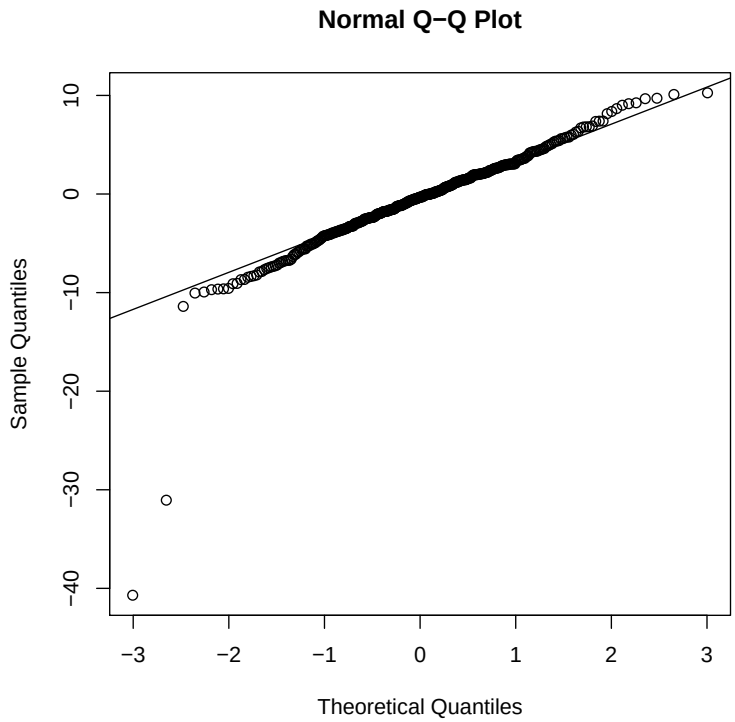

Figure 4.1: QQ norm plot of the treee height prediction errors, in metres.

The plot shows good agreement with our needs in order to invoke the CLT, but also shows that two of the observations differ quite substantially from the predictions. Let's drill down and inspect the trees.

```
> subset(ufc, error.hats.m < -15,
+        select = c(Plot, Tree, Species, Dbh, Height.m, Height.m.p))
```

```
    Plot Tree Species Dbh Height.m Height.m.p
372   67    6      WL 575      3.4   34.45379
411   78    5      WP 667      0.0   40.69514
```

The output shows that the two trees in question have very small heights for their diameters (recall that the diameters are measured in mm). We could quite reasonably exclude them from further consideration, and then feel comfortable with our use of the CLT.

```
> ufc.nice <- subset(ufc, error.hats.m > -15)
```

We use 2 metres as our region of indifference; that is, we'll say that if the average height of the predictions is demonstrably within 2 metres of the average height of the observations, then the null hypothesis should be rejected. Here, we save the test object that the function creates for subsequent examination.

```
> fvs.test <- tost(x = ufc.nice$Height.m.p,
+                  y = ufc.nice$Height.m,
+                  epsilon = 2)
```

The returned object is a list of objects, so we'll dig around it here.

```
> str(fvs.test)
```

```
List of 9
 $ mean.diff: num 0.412
 $ se.diff  : num 0.511
 $ alpha    : num 0.05
 $ ci.diff  : atomic [1:2] -0.429 1.254
  ..- attr(*, "conf.level")= num 0.9
 $ df       : Named num 738
  ..- attr(*, "names")= chr "df"
 $ epsilon  : num 2
 $ result   : chr "rejected"
 $ p.value  : num 0.000982
 $ check.me : atomic [1:2] -1.18 2
  ..- attr(*, "conf.level")= num 0.998
```

The object comprises the following pieces:

1. `mean.diff` is the difference between the sample means,
2. `se.diff` is the estimated standard error of the difference between the sample means,
3. `alpha` is the nominated size of the test,
4. `ci.diff` is the estimated $(1 - \alpha)$ confidence interval of the difference between the population means,
5. `df` is the number of degrees of freedom used for the calculation,
6. `epsilon` is the nominated region of similarity,
7. `result` is the outcome of the hypothesis test — the null hypothesis is either rejected or not rejected,

8. `p.value` reports 1 minus the coverage of the largest possibly interval that will result in rejection of the null hypothesis, and
9. `check.me` reports the largest possibly interval that will result in rejection of the null hypothesis. As much as anything, it is produced to cross-check the reported p-value.

Here, the outcome of the hypothesis test is:

```
> fvs.test$result

[1] "rejected"
```

with p-value

```
> fvs.test$p.value

[1] 0.0009824095
```

We conclude that there is strong evidence to support the use of the FVS height–diameter model in this stand.

The rationale behind the interpretation of the p-value is the same as that that underpins the connection between confidence intervals and traditional (splitting) hypothesis tests: the p-value is 1 minus the coverage of the largest possible interval that includes the null hypothesis value.

4.3 Equivalence plot

Robinson et al. (2005) suggested an extension of the TOST that could be used to identify components of lack-of-fit, and provide an integrated summary of the relationship between two variables along with the necessary information to make inference. For want of a better name it is presently termed an "equivalence plot".

The equivalence plot involves making a scatterplot of the raw data, and imposing graphical images on the plot that summarize two TOST-based tests: a test on the intercept (equivalent to a test on the sample mean) and a test on the slope. The plot is called using the following code:

```
> ufc.ht <- ufc.nice[!is.na(ufc.nice$Height), ]
> equivalence.xyplot(ufc.ht$Height.m ~ ufc.ht$Height.m.p,
+     alpha = 0.05, b0.ii = 1, b1.ii = 0.15, b0.absolute = TRUE,
+     xlab = "Predicted height (m)", ylab = "Measured height (m)")
```

and is shown in Figure 4.2. We are comparing field measurements of tree height against predictions from models published by Wykoff et al. (1982). Here the test *rejects* the null hypothesis of difference from the intercept to 0 and the slope to 1. The mean of the model predictions is close to the mean of the height measurements, and the tall trees are predicted to be taller than the short trees. We can use this model for the population from which the sample came with some confidence.

It is worth noting that the quality of the match between the data and the model here is quite extraordinary, and we would normally not expect the concordance between the model and the test data to clear quite so high a bar.

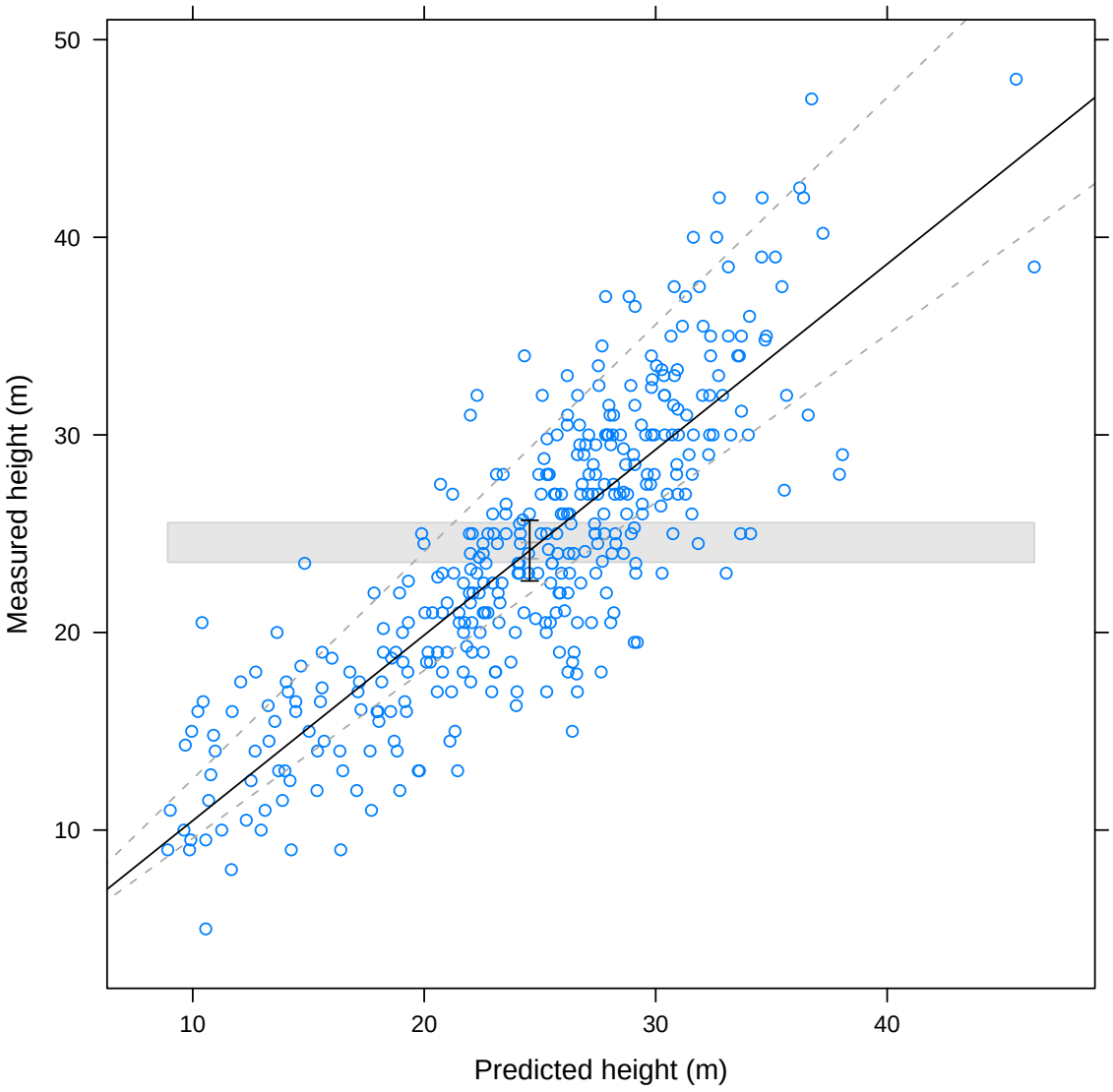

Figure 4.2: *TOST embedded in a regression framework (see Robinson et al., 2005). The TOST for the intercept is carried out by assessing whether the (grey) intercept error bar is inside the grey rectangle, which represents a region of similarity of ±1m of the mean. The TOST for the slope is carried out by assessing whether the (black) slope error bar is inside the dotted rays, which represent a region of similarity of ±15% of the slope. Here, the slope and intercept error bars are coincident, owing to the large sample size. The test rejects the null hypothesis of difference from the intercept to 0 and the slope to 1.*

Bibliography

Bates, D., Maechler, M., 2010. lme4: Linear mixed-effects models using S4 classes. R package version 0.999375-33.
URL http://CRAN.R-project.org/package=lme4

Berger, R. L., Hsu, J. C., 1996. Bioequivalence trials, intersection-union tests and equivalence confidence sets. Statistical Science 11 (4), 283–319.

Bolker, B. M., Brooks, M. E., Clark, C. J., Geange, S. W., Poulsen, J. R., Stevens, M. H. H., White, J.-S. S., 2009. Generalized linear mixed models: a practical guide for ecology and evolution. Trends in Ecology & Evolution 24 (3), 127–135.

Box, G. E. P., 1953. Non-normality and tests on variances. Biometrika 40 (3/4), 318–335.

Daubenmire, R., 1952. Forest vegetation of Northern Idaho and adjacent Washington, and its bearing on concepts of vegetation classification. Ecological Monographs 22, 301–330.

Demidenko, E., Stukel, T. A., 2005. Influence analysis for linear mixed-effects models. Statistics in Medicine 24, 893–909.

Dixon, P. M., Pechmann, J. H. K., 2005. A statistical test to show negligible trend. Ecology 86 (7), 1751–1756.

Fitzmaurice, G., Laird, N., Ware, J., 2004. Applied Longitudinal Analysis. John Wiley & Sons, 506 p.

Gelman, A., Hill, J., 2007. Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press, 625 p.

Laird, N. M., Ware, J. H., 1982. Random-effects models for longitudinal data. Biometrics 38, 963–974.

McBride, G. B., 1999. Equivalence tests can enhance environmental science and management. Australian and New Zealand Journal of Statistics 41 (1), 19–29.

Parkhurst, D. F., 2001. Statistical significance tests: equivalence and reverse tests should reduce misinterpretation. Bioscience 51 (12), 1051–1057.

Pinheiro, J. C., Bates, D. M., 2000. Mixed-effects models in S and Splus. Springer-Verlag, 528 p.

Robinson, A. P., Duursma, R. A., Marshall, J. D., 2005. A regression-based equivalence test for model validation: shifting the burden of proof. Tree Physiology 25, 903–913.

Robinson, A. P., Froese, R. E., 2004. Model validation using equivalence tests. Ecological Modelling 176, 349–358.

Robinson, G. K., 1991. That BLUP is a good thing: The estimation of random effects. Statistical Science 6 (1), 15–32.

Rogers, J. L., Howard, K. I., Vessey, J. T., 1993. Using significance tests to evaluate equivalence between two experimental groups. Psychological Bulletin 113 (3), 553–565.

Sanchez, D., Rachlow, J., Robinson, A., Johnson, T., 2009. Survey indicators for pygmy rabbits: Temporal trends of burrow systems and pellets. The Western North American Naturalist 69 (4), 426–436.

Schabenberger, O., 2005. Mixed model influence diagnostics. In: SUGI 29. SAS Institute, pp. Paper 189–29.

Schabenberger, O., Pierce, F. J., 2002. Contemporary statistical models for the plant and soil sciences. CRC Press, 738 p.

Schuirmann, D. L., 1981. On hypothesis testing to determine if the mean of a normal distribution is contained in a known interval. Biometrics 37, 617.

Stage, A. R., 1963. A mathematical approach to polymorphic site index curves for grand fir. Forest Science 9 (2), 167–180.

Venables, W. N., Ripley, B. D., 2000. S programming. Springer-Verlag, 264 p.

Venables, W. N., Ripley, B. D., 2002. Modern applied statistics with S and Splus, 4th Ed., 4th Edition. Springer-Verlag, 495 p.

Wellek, S., 2003. Testing statistical hypotheses of equivalence. Chapman and Hall/CRC.

Westlake, W. J., 1981. Response to T.B.L. Kirkwood: Bioequivalence testing–a need to rethink. Biometrics 37, 589–594.

Wykoff, W. R., Crookston, N. L., Stage, A. R., 1982. User's guide to the Stand Prognosis model. GTR-INT 133, USDA Forest Service.