Multiple-Table Data in R with the multitable Package

Steven C Walker
Université de Montréal

Guillaume Guénard
Université de Montréal

Pierre Legendre
Université de Montréal

Abstract

Data frames are integral to R. They provide a standard format for passing data to model-fitting and plotting functions, and this standard makes it easier for experienced users to learn new functions that accept data as a single data frame. Still, many data sets do not easily fit into a single data frame; data sets in ecology with a so-called fourth-corner problem provide important examples. Manipulating such inherently multiple-table data using several data frames can result in long and difficult-to-read workflows. We introduce the R **multitable** package to provide new data storage objects called `data.list` objects, which extend the `data.frame` concept to explicitly multiple-table settings. Like data frames, data lists are lists of variables stored as vectors; what is new is that these vectors have dimension attributes that make accessing and manipulating them easier. As `data.list` objects can be coerced to `data.frame` objects, they can be used with all R functions that accept an object that is coercible to a `data.frame`.

Keywords: data organisation, ecology, fourth-corner problem, R.

1. Introduction

The standard data management paradigm in R is based on `data.frame` objects, which are two-dimensional data tables with rows and columns representing replicates (sometimes also called objects) and variables (R Development Core Team 2009). Standard R workflows require that the data to be analysed are organised into a data frame (Chambers and Hastie 1992). Hypotheses about the relationships between variables in the data frame are expressed using `formula` objects. Data frames and formulas are combined by passing them to functions that produce analyses (e.g., plots; fitted models; summary statistics). This framework allows scientists to concentrate on their primary interests—the relationships between variables—without explicit reference to mathematical and algorithmic details. It also provides access to those details, which are required for effective analyses and to develop new methods of analysis within the framework. As new methods are developed, researchers simply pass their data frames to new functions in much the same way they would pass them to older functions.

Research in community ecology—the study of the distribution and abundance of multiple interacting species—sometimes involves data sets that do not easily fit within a single data frame. A common example is the fourth-corner problem (Legendre, Galzin, and Harmelin-Vivien 1997), in which three data tables are to be analysed: a sites-by-species table of abundances or occurrences; a table of environmental variables at each site; and a table of traits for each species (Figure 1). Such data are characterised by a conspicuous (lower-right) 'fourth-

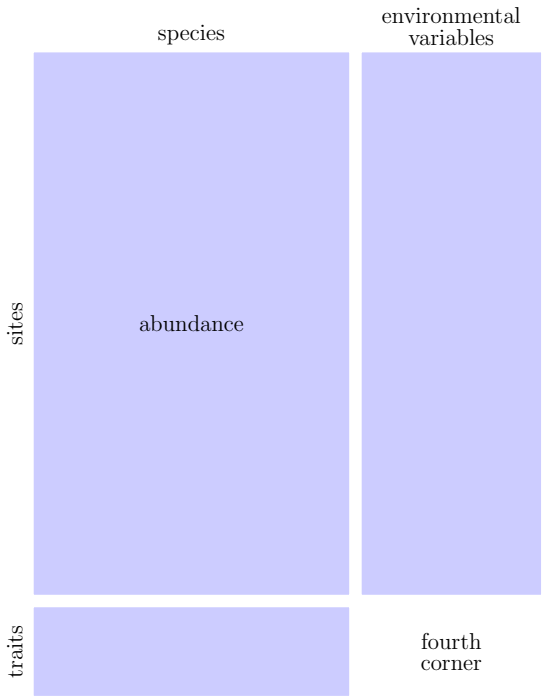


Figure 1: Schematic diagram of a data structure with a fourth-corner problem.

corner', where there are no data. The missing data in the fourth corner are not caused by the usual problems (e.g., broken field equipment; budget restrictions; bad weather; dead subjects), but are part of the study design itself. The fourth-corner problem is a special case of a general 'multiple-table problem', which can be much more complex (e.g., could involve three-dimensional 'cubes' of data, Figure 2). The challenge of analysing such multiple-table data sets in R is that it is not obvious how to organise them into a single `data.frame`, which is required in standard R workflows. Our goal with the R **multitable** package is to provide tools that make analysing multiple-table data sets easier.

One possible solution is to develop new R analysis functions—or new software packages altogether—that are specifically designed to accept several tables as input. There have been several such methods developed in ecology, focusing on data with a fourth-corner problem (Dolédec, Chessel, ter Braak, and Champely 1996; Legendre *et al.* 1997; Dray and Legendre 2008; Pillar and Duarte 2010; Leibold, Economo, and Peres-Neto 2010; Ives and Helmus 2011). However, these methods do not apply to data sets that have other more complex multiple-table data structures (e.g., the zooplankton communities in Lac Croche, which are described in Figure 2; Cantin, Beisner, Gunn, Prairie, and Winter 2011). One approach to such issues would be to develop suites of data analysis functions for each new data structure. But such an approach is less than ideal as it would require that new methods be developed for each new structure, which does not take advantage of the large number of tools developed for standard R workflows (Chambers and Hastie 1992). The **multitable** package provides an alternative approach, by introducing a multiple-table generalisation of data frames—called data lists—which can be analysed with virtually any function that can be used to analyse

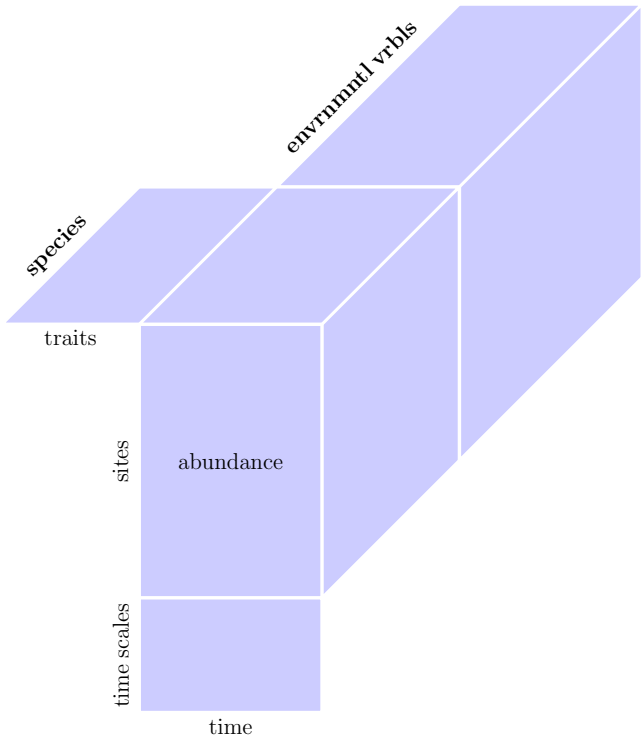


Figure 2: The structure of the Lac Croche zooplankton community data. The abundances of zooplankton species and several environmental variables were measured every two weeks in the summer at various basins (i.e., sites) in the lake over two years—yielding two temporal scales at which sampling took place: week within year and year. In addition, the species were characterised by a suite of traits.

a data frame. Thus, instead of providing new methods of analysis, **multitable** provides new methods of data management and organisation.

How can data lists make data organisation easier? Although practically any data set can be forced into a single `data.frame` by either repeating some of the data or adding missing values, other structures exist that would make a particular data set easier to understand, manipulate, and analyse. Accordingly, we have designed `data.list` objects to provide a richer structure than `data.frame` objects for representing our data 'as we understand them'. This structure furnishes intuitive tools for operating on several data tables simultaneously, thereby saving time and effort that could be better spent thinking about relationships between variables. As we have discussed, there are important advantages to organising data in `data.frame` objects—perhaps the most important advantage being the powerful catalogue of R functions that accept data in such a form. The **multitable** package provides methods for coercing `data.list` objects into `data.frame` objects, thus making standard R tools available to multiple-table data organised as a `data.list` object. In summary, the **multitable** model of data organisation is to manipulate, transform, and extract subsets of our data in `data.list`-form, and then to coerce them into `data.frame`-form when we are ready to pass them to analysis functions (Figure 3).

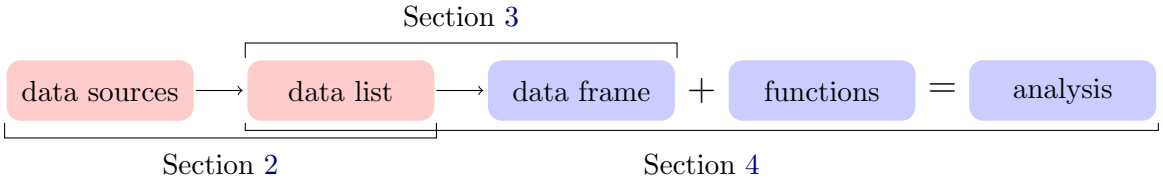
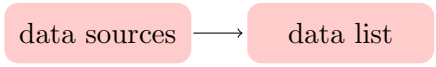


Figure 3: The **multitable** framework for using multiple-table data (in red) in R workflows (in blue). Hyper-linked section numbers indicate portions of the workflow covered by sections in this article. Data lists are used to organise and manipulate multiple-table data as a single R object, even though such data will typically be originally stored in several other sources (e.g., R objects; spreadsheets; text-based data files; database queries). When such data are ready for analysis, they can be coerced into a data frame. With the data in data frame form, they can be analysed and visualised by combining them with any R function that accepts `data.frame` objects.

There are several existing R packages that are designed to make data organisation easier (e.g., **reshape2**; Wickham, 2007). The **mefa** and **mefa4** packages have been developed to organise data with a slight generalisation of the fourth-corner problem (Sólymos 2009); this generalisation permits several community matrices—called segments—with identical dimensions. The **multitable** package has much in common with **mefa**, but there are noticeable differences. For example, **multitable** is designed to handle more general data structures than **mefa** or **mefa4**; in particular, **mefa** is not able to represent the relational structure of the Lac Croche data depicted in Figure 2. On the other hand, **mefa** provides more extensive tools for data summarisation than **multitable** and **mefa4** integrates tools for sparse-matrix computations. We therefore expect **mefa** and **multitable** to often be complementary in practice.

Our purpose is to introduce the **multitable** package, and demonstrate its utility. There are three main organisational sections, each corresponding to a part of Figure 3. Section 2 describes the structure of a simple `data.list` object (2.1); how to create other data lists (2.2); a simple theoretical framework for understanding data lists (2.3); and techniques for manipulating them (2.4, 2.5, 2.6, 2.7, 2.8). Section 3 illustrates the coercion of data lists to data frames. Section 4 illustrates how to use **multitable** in R workflows by summarising (4.2); visualising (4.3); and modelling (4.4, 4.5, 4.6) a real stream fish data set (4.1).

2. Understanding, creating, and manipulating data lists



2.1. The structure of data lists

The **multitable** package comes with a fictitious `data.list`, to illustrate how these objects work.

```
R> library(multitable)
R> data(fake.community)
R> fake.community

abundance:
---------
, , capybara

            2009 2008 1537
midlatitude    4    0    0
subtropical    0   10    0
tropical       8    0    0
equatorial     0    7    0
arctic         0    0    0
subarctic      0    0    0

, , moss

            2009 2008 1537
midlatitude    0    6    0
subtropical    0    0    0
tropical       9    0    0
equatorial     0    3    0
arctic         5    0    0
subarctic      0    0    0

, , vampire

            2009 2008 1537
midlatitude    0    0    0
subtropical    0    0    1
tropical       0    0    0
equatorial     0    0    0
arctic         0    0    0
subarctic      0    0    0
```

```
Replicated along:  || sites || years || species ||


temperature:
-----------
            2009 2008 1537
midlatitude   NA   10   NA
subtropical   25   20   NA
tropical      48   50   NA
equatorial    50   30   NA
arctic       -37  -30   NA
subarctic      3    0   NA
Replicated along:  || sites || years ||


precipitation:
-------------
            2009 2008 1537
midlatitude   NA   20   NA
subtropical   99  100   NA
tropical     149  150   NA
equatorial   199  200   NA
arctic        21   20   NA
subarctic     41   40   NA
Replicated along:  || sites || years ||


body.size:
---------
capybara     moss  vampire
     140       NA      190
Replicated along:  || species ||


metabolic.rate:
--------------
capybara     moss  vampire
      20        5        0
Replicated along:  || species ||


homeotherm:
----------
capybara     moss  vampire
       Y        N        N
Levels: N Y
Replicated along:  || species ||
```

```
REPLICATION DIMENSIONS:
  sites   years species
      6       3       3
```

At first sight, this `data.list` object looks very different from standard `data.frame` objects, but on second look we can see that they are really quite similar. Just like data frames, data lists are composed of a number of variables—in this case, we have six variables (`abundance`; `temperature`; `precipitation`; `body.size`; `metabolic.rate`; and `homeotherm`) each identified in the printed object above by underlined names. The variables in data lists must be printed in this sequential manner, rather than as columns neatly lined up in a data frame, precisely because the variables in multiple-table data sets do not line up neatly; this is the problem **multitable** seeks to address.

Also as with data frames, the replication of variables in data lists are represented as vectors of values. The main difference between the two objects in this regard is that the vectors that represent variables in data lists have `dim` (i.e., dimension) attributes. These `dim` attributes give `data.list` objects further structure. In R, vectors with `dim` attributes are best thought of as matrices and arrays of numbers. For example, the `abundance` variable is replicated along three dimensions (sites; years; and species), and therefore is a three dimensional array of data. This information is displayed after the data whenever a `data.list` object is printed. Some variables are only replicated along two dimensions (e.g., `temperature` and `precipitation`) and others only have one dimension (e.g., `body.size`; `metabolic.rate`; and `homeotherm`).

Importantly however, although the variables are not replicated along all of the same dimensions, they do share dimensions; and it is this dimension sharing that allows us to relate variables to each other. To appreciate the dimension sharing of this example, we can use the `summary` method for `data.list` objects:

```
R> summary(fake.community)
```

```
        abundance temperature precipitation body.size
sites        TRUE        TRUE          TRUE     FALSE
years        TRUE        TRUE          TRUE     FALSE
species      TRUE       FALSE         FALSE      TRUE
        metabolic.rate homeotherm
sites            FALSE      FALSE
years            FALSE      FALSE
species           TRUE       TRUE
```

This method returns a logical matrix with dimensions of replication as rows and variables as columns. A value of `TRUE` appears in cells corresponding to variables that are replicated along a particular dimension, and a value of `FALSE` appears otherwise. We can see that the `sites` and `years` dimensions relate `abundance`, `temperature`, and `precipitation`; whereas, the `species` dimension relates `abundance`, `body.size`, `metabolic rate`, and `homeotherm`.

Note that some `FALSE` entries are biophysical necessities, whereas others are properties of the study design. For example, suppose that later in the study, the researchers decided that it

was necessary to get some idea of the spatial variation in metabolic rates. It would then be possible to measure metabolic rates of the species at different sites, thereby changing the `FALSE` associated with the metabolic rate-sites cell to a `TRUE`. To the contrary, it is both physically and logically impossible to measure the precipitation of a species, so this `FALSE` is mandatory.

2.2. How data lists are made

Although there are several ways to create data lists, one way in particular provides a simple framework for understanding the difference between variables and dimensions of replication—an important distinction to understand in order to use **multitable** effectively.

Consider the following data frame of species abundances counted at various sites.

```
R> abundance
```

```
        sites  species abundance
1 midlatitude capybara         4
2 subtropical capybara        10
3    tropical capybara         8
4  equatorial capybara         7
5      arctic     moss         5
6 midlatitude     moss         6
7    tropical     moss         9
8  equatorial     moss         3
9 subtropical  vampire         1
```

We have six sites and three species, but each species is not present at each site and so there are missing site-species combinations.

Related to this `abundance` data frame we have a data frame of environmental variables at each site and a data frame of traits for each species.

```
R> environment
```

```
        sites temperature precipitation
1   subarctic           0            40
2 midlatitude          10            20
3 subtropical          20           100
4    tropical          50           150
5  equatorial          30           200
```

```
R> trait
```

```
   species body.size metabolic.rate
1 capybara       140             20
2     moss         5              5
3  vampire       190              0
```

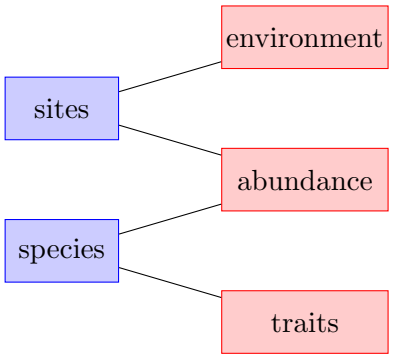


Figure 4: Bipartite graph of the multiple-table structure of data with a standard fourth-corner structure (Figure 1). Dimensions of replication are in blue (on the left) and tables are in red (on the right).

To make things interesting to scientists with real data, we assume that our environmental data are missing from the arctic site (perhaps because it is too remote to go there and make measurements).

The three data frames are related because they share two columns: sites and species. The specific pattern of sharing for these data can be illustrated with a bipartite graph (i.e., matching diagram; Figure 4). Columns that are shared between data frames are called *dimensions of replication* and those that are not are called *variables*. The reason for this terminology is that in standard single-table statistical settings, we are able to relate variables because they are replicated along some common dimension. For example, one can relate pH and temperature if they are both replicated along the same set of lakes. Similarly, we can relate the variables in several tables together if they share columns (i.e., dimensions of replication).

To create a data list out of these data frames we use the data list cast, `dlcast`, function from **multitable**, which was inspired by the `acast` function in the **reshape2** package (Wickham 2007).

```
R> l <- list(abundance, environment, trait)
R> dl <- dlcast(l, fill = c(0, NA, NA))
R> summary(dl)
```

```
         abundance temperature precipitation body.size metabolic.rate
sites         TRUE        TRUE          TRUE     FALSE          FALSE
species       TRUE       FALSE         FALSE      TRUE           TRUE
```

```
R> dl
```

```
abundance:
---------
             capybara moss vampire
arctic              0    5       0
equatorial          7    3       0
midlatitude         4    6       0
```

```
subtropical       10    0       1
tropical           8    9       0
subarctic          0    0       0
Replicated along:  || sites || species ||


temperature:
-----------
     arctic  equatorial midlatitude subtropical    tropical   subarctic
         NA          30          10          20          50           0
Replicated along:  || sites ||


precipitation:
-------------
     arctic  equatorial midlatitude subtropical    tropical   subarctic
         NA         200          20         100         150          40
Replicated along:  || sites ||


body.size:
---------
capybara     moss  vampire
     140        5      190
Replicated along:  || species ||


metabolic.rate:
--------------
capybara     moss  vampire
      20        5        0
Replicated along:  || species ||


REPLICATION DIMENSIONS:
  sites species
      6       3
```

This `dlcast` function takes one mandatory argument: a list of data frames to be combined into a data list. The optional `fill` argument accepts a vector with one element for each data frame, giving the value with which to fill in any structural missing values. This argument is useful because we can both (1) fill missing abundances with zeros because those site-species combinations were not observed and (2) fill missing traits and environmental variables with `NA` values.

Making data lists out of 'wide format' data

Researchers will often have text files or spreadsheets of data that are not stored in the same format as the three data frames in the previous example. These three data frames have two types of columns—some columns represent dimensions of replication and others represent variables. This data storage format is sometimes called 'long format' (see `?reshape`), because more sampling results in a lengthening of the data (i.e., the addition of rows) without any widening (i.e., the addition of columns). In contrast, it is common in community ecology for example to store abundance data as spreadsheets with sites as rows and species as columns (e.g., as in Figure 1). Such a data storage format is often called 'wide format', because more sampling may result in a widening of the data (e.g., more columns are required as further sampling reveals a greater diversity of species). Fortunately, the **multitable** package provides tools for reading data stored in a variety of different formats into a data list. The `as.data.list`, `data.list`, `variable`, `variableGroup`, `read.multitable`, and `read.fourthcorner` functions are all alternatives to `dlcast` for creating data lists.

The `variable` function provides a convenient way to add data stored in wide format to an existing data list. Consider for example, a matrix `allele` containing the frequencies of a particular allele (i.e., alternate form of a gene) for each species at each site,

```
R> allele
```

```
            capybara moss vampire
arctic            NA  0.0      NA
equatorial       0.4  0.0      NA
midlatitude      0.0  0.0      NA
subtropical      0.1   NA       0
tropical         0.0  0.2      NA
subarctic         NA   NA      NA
```

Note that this `allele` variable is in wide format, because additional sampling could lead to a widening of the matrix. One can add such wide data to an existing data list using the `+` operator and the `variable` function,

```
R> dl.with.allele <- dl + variable(allele, c("sites", "species"))
```

The `variable` function creates a data list with a single variable (i.e., `allele`) and the `+` operator 'adds' it to `dl`. Notice that the `+` operator here does not have its usual arithmetic meaning, but instead means 'merge two data lists together'. The two arguments of `variable` give the data to be converted into a variable (the vector `allele` in this case) and identifies the dimensions along which the variable is replicated (which are `"sites"` and `"species"` in this case). The result of this addition of two data lists contains one more variable than `dl`,

```
R> summary(dl.with.allele)
```

```
        abundance temperature precipitation body.size metabolic.rate allele
sites        TRUE        TRUE          TRUE     FALSE          FALSE   TRUE
species      TRUE       FALSE         FALSE      TRUE           TRUE   TRUE
```

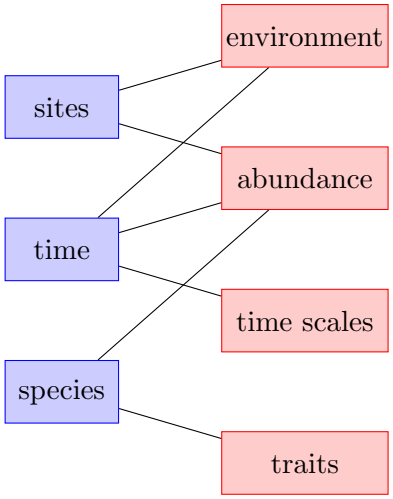


Figure 5: Bipartite graph of the Lac Croche data in Figure 2. Dimensions of replication are in blue (on the left) and tables are in red (on the right).

2.3. Multiple-table concepts

The **multitable** package is based on a distinction between dimensions of replication and variables. One benefit of this distinction is that it provides a common framework for understanding both simple and more complex multiple-table data structures. In particular, the framework allows one to visualise the structure of complex data using bipartite graphs; for example the Lac Croche zooplankton community data (Figure 2) (Cantin *et al.* 2011) has a structure given by Figure 5. To store these data in a format amenable to `dlcast` (i.e., 'long format'), we would create one data frame for each of the groups of variables (red boxes on the right) and add a column for each dimension of replication (blue boxes on the left) associated with those variables.

Visualising the structure of data in this way will help to clarify how it should be both organised and analysed. One of the central themes of **multitable** is that thinking about data organisation goes a long way towards clarifying how analysis should proceed. The names of what we store as variables will appear in formula objects, so that we can study the relationships between these variables. On the other hand, the information that we have for inferring these relationships will come from what we store as dimensions of replication. In single-table settings we keep these two elements of data analysis separate by storing variables as columns and replicates as rows in a `data.frame`. The `data.list` concept is very similar except that replication now has a dimensionality, which allows for the storage of more complex data structures. The basic distinction between variables and replicates guides analysis in multiple-table settings just as it does in single-table settings.

To store data as a `data.frame` object, all variables must have the same length. This requirement ensures that the variables can be related to each other along the single dimension of replication defined by the rows of the data frame. The analogous requirement for storing data in a `data.list` object is that at least one variable must be replicated along all of the dimensions present in the data set. This requirement ensures two important properties: (1) every variable can be related to at least one other variable along at least one dimension of

replication and (2) at least one variable will be relatable to all other variables, a property that is necessary for a response variable. For example, the Lac Croche data (Figure 5) meets this requirement because the abundance variable is replicated along all three dimensions of replication. Therefore, all other variables must at least share one dimension with the abundance variable (e.g., the traits and abundance share the species dimension).

2.4. Subscripting data lists

When analysing data, it is often of interest to extract a part of the data. For example, examining the data suggests that 1537 might have been an outlying year relative to 2008 and 2009. We can exclude data from 1537 just as we would with a single R array:

```
R> fake.community[,c("2008","2009"),]

abundance:
---------
, , capybara

            2008 2009
midlatitude    0    4
subtropical   10    0
tropical       0    8
equatorial     7    0
arctic         0    0
subarctic      0    0

, , moss

            2008 2009
midlatitude    6    0
subtropical    0    0
tropical       0    9
equatorial     3    0
arctic         0    5
subarctic      0    0

, , vampire

            2008 2009
midlatitude    0    0
subtropical    0    0
tropical       0    0
equatorial     0    0
arctic         0    0
subarctic      0    0

Replicated along:  || sites || years || species ||
```

```
temperature:
-----------
            2008 2009
midlatitude   10   NA
subtropical   20   25
tropical      50   48
equatorial    30   50
arctic       -30  -37
subarctic      0    3
Replicated along:  || sites || years ||


precipitation:
-------------
            2008 2009
midlatitude   20   NA
subtropical  100   99
tropical     150  149
equatorial   200  199
arctic        20   21
subarctic     40   41
Replicated along:  || sites || years ||


body.size:
---------
capybara     moss  vampire
     140       NA      190
Replicated along:  || species ||


metabolic.rate:
--------------
capybara     moss  vampire
      20        5        0
Replicated along:  || species ||


homeotherm:
----------
capybara     moss  vampire
       Y        N        N
Levels: N Y
Replicated along:  || species ||
```

```
REPLICATION DIMENSIONS:
  sites   years species
      6       2       3
```

This command returns the same data list of variables but without the data from 1537. Note that every variable replicated along the `years` dimension is subscripted appropriately, while variables that are not replicated along this dimension are unchanged. As another example, perhaps we want all of the data from the first three sites, in 1537, for the first species (i.e., capybara). The following line would produce such a data list:

```
R> fake.community[1:3, "1537", c(TRUE, FALSE, FALSE)]
```

This example illustrates that data list subscripting can be done with integers (e.g., `1:3`), character strings (e.g., `"1537"`), and logical vectors (e.g., `c(TRUE, FALSE, FALSE)`).

The previous subscripting examples operated on all variables in the data list simultaneously. However, it is often useful to be able to extract subsets of the variables themselves. Such variable extraction can be done by passing a single subscripting vector that refers to variables instead of dimensions of replication,

```
R> fake.community[c("temperature", "precipitation")]
```

```
temperature:
-----------
             2009 2008 1537
midlatitude    NA   10   NA
subtropical    25   20   NA
tropical       48   50   NA
equatorial     50   30   NA
arctic        -37  -30   NA
subarctic       3    0   NA
Replicated along:  || sites || years ||


precipitation:
-------------
             2009 2008 1537
midlatitude    NA   20   NA
subtropical    99  100   NA
tropical      149  150   NA
equatorial    199  200   NA
arctic         21   20   NA
subarctic      41   40   NA
Replicated along:  || sites || years ||


REPLICATION DIMENSIONS:
```

```
sites years
    6     3
```

If subscripting results in a data list with only a single dimension of replication, then the default behaviour is to coerce to a data frame; for example,

```
R> fake.community[5:6]
```

```
         metabolic.rate homeotherm
capybara             20          Y
moss                  5          N
vampire               0          N
```

To suppress this behaviour, use,

```
R> fake.community[5:6, drop = FALSE]
```

```
metabolic.rate:
--------------
capybara     moss  vampire
      20        5        0
Replicated along:  || species ||


homeotherm:
----------
capybara     moss  vampire
       Y        N        N
Levels: N Y
Replicated along:  || species ||


REPLICATION DIMENSIONS:
species
      3
```

The `drop` argument refers to whether or not the the dimensional structure of the data list should be dropped, because a data frame is sufficient to organise the results of a data list with only a single dimension of replication.

It is important to distinguish between the two ways in which data lists can be subscripted: extraction of subsets of (1) dimensions of replication and (2) variables. We refer to these two subscripting techniques as array-like and list-like—array-like because when R arrays are subscripted, subsets of dimensions are extracted; and list-like because when R lists are subscripted, subsets of variables are extracted. Both the array-like and list-like subscripting functions are designed to behave as similarly as possible to standard **base** R array and list subscripting. See the `Extract.data.list` help file for more details.

2.5. Assigning new values to variables in data lists

Often we need to alter the values of variables before passing data frames to functions. This is easily done with variables in data lists as well. For example. we note that `fake.community` has a lot of missing values. Suppose that these missing observations were made in a subsequent sampling campaign. We can replace these missing values with the new observations using the standard logic of R replacement.

```
R> fake.community$precipitation[is.na(fake.community$precipitation)] <-
+   c(30, 5, 50, 75, 50, 2, 7)
```

2.6. Creating transformed variables in data lists

Another common task in data analysis is to create new variables that are transformed versions of older variables in the data set. For example, suppose we want to make a log transformation of the abundance data. If `fake.community` was a data frame, one could create a new logged version of `abundance` with the following command,

```
R> fake.community$log.precipitation <- log(fake.community$precipitation)
```

However, because `fake.community` is a data list rather than a data frame, this command results in the following error: `can't add variables this way...try using [[ instead of $...and don't forget to specify match.dimids or shape`. The reason why this command works for data frames but not data lists, is that the dimensional structure of data lists must also be specified for this new variable. A simple approach to such a specification is to use the `shape` argument, which identifies an existing variable with the same shape (i.e., dimensions of replication) as the new variable,

```
R> fake.community[['log.precipitation', shape = 'precipitation']] <-
+   log(fake.community$precipitation)
R> fake.community['log.precipitation']

log.precipitation:
-----------------
                2009     2008      1537
midlatitude 3.401197 2.995732 1.6094379
subtropical 4.595120 4.605170 3.9120230
tropical    5.003946 5.010635 4.3174881
equatorial  5.293305 5.298317 3.9120230
arctic      3.044522 2.995732 0.6931472
subarctic   3.713572 3.688879 1.9459101
Replicated along:  || sites || years ||


REPLICATION DIMENSIONS:
sites years
    6     3
```

2.7. Creating variables to identify replicates

It is often useful to construct variables that identify replicates. For example, if one of the dimensions of replication of a data list varies across spatial locations (e.g., sites), it is often of interest to examine potential site effects. A site ID variable will be required to construct models with such a site effect. Therefore, `site` must be considered both a dimension of replication and a variable. The `dims_to_vars` function in the **multitable** package is a convenience function for creating variables out of dimensions of replication,

```
R> fake.community <- dims_to_vars(fake.community)
R> summary(fake.community)
```

```
        abundance temperature precipitation body.size metabolic.rate homeotherm
sites        TRUE        TRUE          TRUE     FALSE          FALSE      FALSE
years        TRUE        TRUE          TRUE     FALSE          FALSE      FALSE
species      TRUE       FALSE         FALSE      TRUE           TRUE       TRUE
        log.precipitation sites years species
sites                TRUE  TRUE FALSE   FALSE
years                TRUE FALSE  TRUE   FALSE
species             FALSE FALSE FALSE    TRUE
```

Note that this manipulated data list now contains variables associated with the three dimensions of replication (i.e., `sites`; `years`; and `species`). For example, here is the variable associated with the `species` dimension,

```
R> fake.community['species', drop = FALSE]
```

```
species:
-------
capybara     moss  vampire
capybara     moss  vampire
Levels: capybara moss vampire
Replicated along:  || species ||


REPLICATION DIMENSIONS:
species
      3
```

2.8. Melting and recasting data lists

We now describe a fairly advanced technique. Occasionally it is difficult to manipulate a data list into the desired form. For example, note that an identical set of variables are replicated along both `sites` and `years` in the following subset of the `fake.community` data,

```
R> data(fake.community)
R> fake.community <- fake.community[1:3, 1:2, 1:2][1:4]
R> summary(fake.community)
```

```
        abundance temperature precipitation body.size
sites        TRUE        TRUE          TRUE     FALSE
years        TRUE        TRUE          TRUE     FALSE
species      TRUE       FALSE         FALSE      TRUE
```

When two dimensions share the same variables it is always possible to collapse them into a single dimension, without loss of information. In other words, instead of three dimensions (e.g., `sites`; `years`; and `species`) one could have two dimensions (e.g., `sites.years` and `species`). Below in Section 4.6 we will give an example of where it would be useful to collapse dimensions in this way. Here we show how to actually do the collapsing.

In future versions of **multitable** there may be a `collapse` function. In the meantime, we can use a **multitable** implementation of the melt-cast approach to complex data manipulations that has become popular via the **reshape2** package (Wickham 2007). The basic principal of **reshape2** is that it is often difficult to go from one type of structure to another—rather it can be easier to 'melt' away the structure and then 'cast' the molten data into a new form.

The `dlmelt` function in **multitable** allows one to remove structure from data lists. This function is similar to the `melt` function in the **reshape2** package, which converts many different kinds of data into 'long' (i.e., database-like) format (see Section 2.2). However, it is not possible to melt a data list into a single long-format data frame without repeating many of its entries—a result of the multiple-table structure of data lists. Therefore, `dlmelt` first separates variables into groups such that all variables in a group can be stored together in a single data frame. The common feature that variables in such a group share is that they are all replicated along the same dimensions. For example,

```
R> dlm <- dlmelt(fake.community)
```

This `dlm` object is a list of three data frames,

```
R> summary(dlm)
```

```
                    Length Class      Mode
sites.years.species 4      data.frame list
sites.years         4      data.frame list
species             2      data.frame list
```

each containing the variables replicated along the dimensions indicated by the names of the data frames. The first two lines of the three data frames are,

```
R> lapply(dlm, head, n = 2)
```

```
$sites.years.species
                         abundance       sites years  species
midlatitude.2009.capybara         4 midlatitude  2009 capybara
subtropical.2009.capybara         0 subtropical  2009 capybara

$sites.years
                temperature precipitation       sites years
```

```
midlatitude.2009          NA            NA midlatitude  2009
subtropical.2009          25            99 subtropical  2009

$species
         body.size  species
capybara       140 capybara
moss            NA     moss
```

The columns of each of these data frames give both variables and dimensions of replication.

We could recast this melted multiple-table data set and get the original data list back, with the following command,

```
R> dlcast(dlm)
```

But what we want is to collapse the `sites` and `years` dimensions together before recasting. To do this, we need to make new columns representing the collapsed dimension of replication, and delete the columns associated with original un-collapsed dimensions. These new columns can be created with the `interaction` and `within` functions in **base** R,

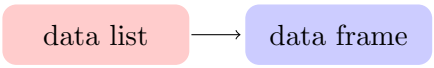
```
R> dlm$sites.years.species <- within(dlm$sites.years.species, {
+   sites.years <- interaction(sites, years)
+   sites <- years <- NULL
+ })
R> dlm$sites.years <- within(dlm$sites.years, {
+   sites.years <- interaction(sites, years)
+   sites <- years <- NULL
+ })
```

Recasting can then be applied to produce the reshaped data list,

```
R> dl <- dlcast(dlm)
R> summary(dl)

            abundance temperature precipitation body.size
species          TRUE       FALSE         FALSE      TRUE
sites.years      TRUE        TRUE          TRUE     FALSE
```

3. Coercing data lists to data frames



A pivotal point in any workflow using **multitable** is the coercion of a data list to a data frame—this is the point at which the variety of R tools for single-table data, become available to multiple-table data. Because this coercion is so pivotal, the syntax required to execute it is as simple as possible,

```
R> data(fake.community)
R> fake.community.df <- as.data.frame(fake.community)
```

The resulting data frame contains one column for each variable and one row for each combination of replicates across the three dimensions of replication; Table 1 shows this data frame, without the last column so that it can fit on one page. Notice that the row names are automatically generated to be informative about the dimensions of replication that have been collapsed into a single dimension. Unlike the corresponding data list object, the data frame has redundancy. For example, because `body.size` is only replicated along `species` there are only three unique `body.size` values, one for each of the three species. These three values are repeated so that all of the variables can be stored side-by-side in a single data frame.

3.1. Faster iterative coercion of data lists to data frames

On occasion, one may wish to iteratively coerce a sequence of data lists to data frames. For example, in a randomisation test one might loop over a number of random subscripts of a data list (Section 4.5). One may find that such an iterative procedure takes too long to run. Fortunately, we can exploit the fact that each replicated data list has the same relational structure (i.e., the same replication dimensions and variables) to reduce computation times. In particular, much of the computational effort involved in coercing data lists to data frames can be done once for all data lists with the same structure; we refer to this initial computation as 'molding'.

Molding begins by taking a 'mold' of the original data list,

```
R> data(fake.community)
R> fc.mold <- data.list.mold(fake.community)
```

With such a mold, the `as.data.frame` operation can be computed much faster than without one. In particular, this command,

```
R> as.data.frame(fake.community, mold = fc.mold)
```

is faster than this one,

```
R> as.data.frame(fake.community)
```

To demonstrate the computational savings of molding, consider the following two simple functions that each coerce the `fake.community` data list to a data frame one hundred times.

```
R> with_molding <- function(){
+   fake.community.mold <- data.list.mold(fake.community)
+   for(i in 1:100)
+     as.data.frame(fake.community, mold = fake.community.mold)
+ }
R> without_molding <- function(){
+   for(i in 1:100)
+     as.data.frame(fake.community)
+ }
```

Table 1: The `fake.community` `data.list` object that has been coerced into a `data.frame`. The final column (`homeotherm`) is omitted so that only one page is required.

	abundance	temperature	precipitation	body.size	metabolic.rate
midlatitude.2009.capybara	4	NA	NA	140	20
subtropical.2009.capybara	0	25	99	140	20
tropical.2009.capybara	8	48	149	140	20
equatorial.2009.capybara	0	50	199	140	20
arctic.2009.capybara	0	-37	21	140	20
subarctic.2009.capybara	0	3	41	140	20
midlatitude.2008.capybara	0	10	20	140	20
subtropical.2008.capybara	10	20	100	140	20
tropical.2008.capybara	0	50	150	140	20
equatorial.2008.capybara	7	30	200	140	20
arctic.2008.capybara	0	-30	20	140	20
subarctic.2008.capybara	0	0	40	140	20
midlatitude.1537.capybara	0	NA	NA	140	20
subtropical.1537.capybara	0	NA	NA	140	20
tropical.1537.capybara	0	NA	NA	140	20
equatorial.1537.capybara	0	NA	NA	140	20
arctic.1537.capybara	0	NA	NA	140	20
subarctic.1537.capybara	0	NA	NA	140	20
midlatitude.2009.moss	0	NA	NA	NA	5
subtropical.2009.moss	0	25	99	NA	5
tropical.2009.moss	9	48	149	NA	5
equatorial.2009.moss	0	50	199	NA	5
arctic.2009.moss	5	-37	21	NA	5
subarctic.2009.moss	0	3	41	NA	5
midlatitude.2008.moss	6	10	20	NA	5
subtropical.2008.moss	0	20	100	NA	5
tropical.2008.moss	0	50	150	NA	5
equatorial.2008.moss	3	30	200	NA	5
arctic.2008.moss	0	-30	20	NA	5
subarctic.2008.moss	0	0	40	NA	5
midlatitude.1537.moss	0	NA	NA	NA	5
subtropical.1537.moss	0	NA	NA	NA	5
tropical.1537.moss	0	NA	NA	NA	5
equatorial.1537.moss	0	NA	NA	NA	5
arctic.1537.moss	0	NA	NA	NA	5
subarctic.1537.moss	0	NA	NA	NA	5
midlatitude.2009.vampire	0	NA	NA	190	0
subtropical.2009.vampire	0	25	99	190	0
tropical.2009.vampire	0	48	149	190	0
equatorial.2009.vampire	0	50	199	190	0
arctic.2009.vampire	0	-37	21	190	0
subarctic.2009.vampire	0	3	41	190	0
midlatitude.2008.vampire	0	10	20	190	0
subtropical.2008.vampire	0	20	100	190	0
tropical.2008.vampire	0	50	150	190	0
equatorial.2008.vampire	0	30	200	190	0
arctic.2008.vampire	0	-30	20	190	0
subarctic.2008.vampire	0	0	40	190	0
midlatitude.1537.vampire	0	NA	NA	190	0
subtropical.1537.vampire	1	NA	NA	190	0
tropical.1537.vampire	0	NA	NA	190	0
equatorial.1537.vampire	0	NA	NA	190	0
arctic.1537.vampire	0	NA	NA	190	0
subarctic.1537.vampire	0	NA	NA	190	0

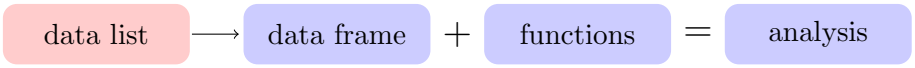
The only difference between these functions is that a mold is created outside of the loop, and therefore should be faster. This first function is indeed faster as the following timing shows,

```
R> library(rbenchmark)
R> benchmark(with_molding(), without_molding(),
+   replications = 10,
+   columns = c("test", "replications", "relative"))
```

```
               test replications relative
1    with_molding()           10 1.000000
2 without_molding()           10 1.299129
```

This output indicates that the function without molding takes ≈ 1.5 times as long. For more on this technique see the help file for `data.list.mold` in the **multitable** package, and the randomisation test in Section 4.5.

4. Analysing data lists



4.1. Stream fish data

Now that we have described how data lists are created and manipulated using a simple example, we move on to analysing a small yet real data set on stream fish communities in Texas (Higgins 2009) that comes pre-loaded with **multitable**,

```
R> data(higgins)
```

The relational structure of this data set is given in Figure 6, and can be accessed in more detail using the `summary` and `dim` functions,

```
R> summary(higgins)
```

```
        abundance width  temp depth velocity substrate habitat
species      TRUE FALSE FALSE FALSE    FALSE     FALSE   FALSE
seasons      TRUE  TRUE  TRUE  TRUE     TRUE      TRUE    TRUE
rivers       TRUE  TRUE  TRUE  TRUE     TRUE      TRUE    TRUE
        trophic life.history
species    TRUE         TRUE
seasons   FALSE        FALSE
rivers    FALSE        FALSE
```

```
R> dim(higgins)
```

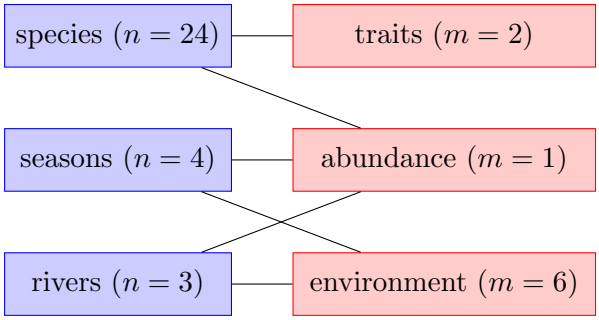


Figure 6: Bipartite graph of the `higgins` stream fish data. Dimensions of replication are in blue (on the left) and tables are in red (on the right). Sample sizes along each dimension of replication are given as n; and numbers of variables in each variable group are given as m.

```
species seasons  rivers
     24       4       3
```

One 100-m stretch of each of three rivers (tributaries of the Colorado River) were sampled once in each of the four seasons. Abundances of 24 fish species were measured as numbers of individuals captured by electro-fishing; we abbreviate species names following Higgins (2009). Two categorical traits (trophic status and life history) were obtained from the stream fish literature to characterise each of the 24 species. Trophic status characterises species' diet and is described by four levels in this data set (herbivore; omnivore; insectivore; and piscivore). Life history describes the strategy by which a species' individuals allocate limited resources amongst themselves, and is based on a three-way classification of strategies (equilibrium, or large investment in relatively few individual offspring; opportunistic, or small size and rapid maturation; and periodic, or pulsed production of large numbers of small offspring); some species are also described as intermediate between two categories. Six environmental variables (stream width (m), temperature (°C), depth (cm), and velocity (m s^{-1}), and two dimensionless measures of river bed substrate and fish habitat) were measured at each river in each season.

4.2. Marginal summaries

A common task in data exploration is to compute marginal summaries (e.g., means and quantiles). In R this is commonly done using the `apply` family of functions or more recently with packages such as **plyr** (Wickham 2011). In general these functions 'apply' another function to various parts of an R object. Here we demonstrate how to use existing apply functions to summarise data lists and introduce a new function: data list apply, `dlapply`.

The `dlapply` function is designed to be intuitive for anyone familiar with the standard `apply` function in **base** R—the arguments for the two functions are identical,

- `apply(X, MARGIN, FUN, ...)`
- `dlapply(X, MARGIN, FUN, ...)`

The `dlapply` function attempts to use `apply` on each variable in a data list, in order to return another data list replicated along the marginal dimensions of replication (i.e., the dimensions

specified in `MARGIN`). However, the rich structure of data lists often makes it impossible to completely marginalise in this way. Still, `dlapply` is extremely useful because it not only marginalises as many of the variables in a data list as possible, but also provides informative messages about why some variables are unable to be marginalised, allowing our thoughts to be centred on data analysis rather than the details of marginalisation. For example, to obtain the median values of the variables in `higgins` for each species, we set `MARGIN = 1` because `species` is the first dimension and set `FUN = median`,

```
R> dlapply(higgins, 1, median)
```

```
omiting width because it is not replicated along MARGIN
omiting temp because it is not replicated along MARGIN
omiting depth because it is not replicated along MARGIN
omiting velocity because it is not replicated along MARGIN
omiting substrate because it is not replicated along MARGIN
omiting habitat because it is not replicated along MARGIN
omiting trophic because of the following error:
Error in median.default(newX[, i], ...)  :  need numeric data
omiting life.history because of the following error:
Error in median.default(newX[, i], ...)  :  need numeric data

abundance:
---------
amna caan caca cicy cyve dini etle etsp gaaf icpu leau lecy legu lema
 0.0  3.5  0.0  0.0 34.5 10.0 11.5  6.0 41.0  1.5 12.5  9.5  0.0 10.5
leme mido mipu misa moco noam nost peca pivi pyol
14.0  0.0  1.0  0.0  0.5  1.0  7.0  6.0  0.0  0.0
Replicated along:  || species ||


REPLICATION DIMENSIONS:
species
     24
```

The species median abundances are given as a single-variable data list and the messages in red explain why the other variables did not marginalise. All of the environmental variables are not replicated along `species` and so it is impossible to find species medians for them. While the `trophic` and `life.history` traits are replicated along `species`, their medians cannot be computed because they are factors. Note that these messages are *not* errors, even though the word 'error' frequently appears in them; reporting on which variables were unable to be marginalised because of an error in the function, `FUN`, being applied is part of the normal behaviour of `dlapply`.

Sometimes results are more visually pleasing when `dlapply` is wrapped in a call to `as.data.frame`, particularly when the resulting data list is large,

```
R> as.data.frame(dlapply(higgins, c(2, 3), median))
```

```
omiting trophic because it is not replicated along MARGIN
omiting life.history because it is not replicated along MARGIN
```

```
                  abundance width  temp depth velocity substrate habitat
spring.pedernales         1.0 20.43 23.32 47.62     3.00      3.68    2.70
summer.pedernales         0.0 19.19 28.42 34.28     2.88      3.72    2.49
fall.pedernales           8.0 21.88 15.00 36.68     4.14      3.83    2.57
winter.pedernales         0.5 22.37 15.28 35.32     3.13      3.68    2.49
spring.san.saba           0.5 30.56 20.32 32.38     1.20      4.78    2.71
summer.san.saba           1.0 23.04 26.25 20.86     0.00      4.63    2.64
fall.san.saba             4.0 35.12 11.73 29.43     0.00      4.63    2.49
winter.san.saba           5.5 37.66 11.44 34.21     1.37      4.64    2.37
spring.south.llano        2.5 14.45 21.54 46.64     0.25      3.34    2.72
summer.south.llano        1.0 11.91 25.49 34.42     0.00      2.93    2.83
fall.south.llano         10.0 13.05 11.42 37.90     0.00      3.31    2.58
winter.south.llano        6.0 14.28 12.18 45.90     0.05      2.94    2.58
```

The result here gives the median values of several variables for each season-river combination. One clear pattern in the median `abundance`s is that the three rivers have more fish in fall and winter than in spring and summer.

While `median` returns a single value, we often want to apply a function that returns several values in a vector (e.g., the `quantile` function). This is easily done with `dlapply` (omitting several variables to save space),

```
R> dlapply(higgins[1:2], 3, quantile)

abundance:
---------
     pedernales san.saba south.llano
0%            0        0         0.0
25%           0        0         0.0
50%           1        2         2.5
75%           9       13        13.5
100%        187      529       212.0
Replicated along:  || quantile || rivers ||


width:
-----
     pedernales san.saba south.llano
0%      19.1900   23.040     11.9100
25%     20.1200   28.680     12.7650
50%     21.1550   32.840     13.6650
75%     22.0025   35.755     14.3225
100%    22.3700   37.660     14.4500
Replicated along:  || quantile || rivers ||


REPLICATION DIMENSIONS:
quantile   rivers
       5        3
```

We see that when the length of the return value of `FUN` is greater than one, a new dimension of replication is created with the name of `FUN` to store this additional information. However, if this new dimension of replication is not the same length for each variable, then a data list cannot be created that contains all of the results. For example, the following command results in an error,

```
R> dlapply(higgins[1:2], 3, summary)
```

```
Error in dlapply(higgins[c(1, 2, 8)], 3, summary) :
results could not be combined into a data list
```

The reason for this error is that the `summary` function does not summarise each type of object in the same way. In this example, summary is applied along a margin of a numerical array, a numerical matrix, and a factor; the output in each of these cases is too different to be combined back into a single data list. However, data lists are also standard R lists,

```
R> is.list(higgins) && is.data.list(higgins)
```

```
[1] TRUE
```

Therefore, functions such as `summary` can be applied to complex data lists using the standard list apply function, `lapply`, in **base** R,

```
R> lapply(higgins[c('abundance', 'width', 'trophic')], summary)
```

```
$abundance
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    1.00   17.18   12.25  529.00

$width
   pedernales       san.saba      south.llano
 Min.   :19.19   Min.   :23.04   Min.   :11.91
 1st Qu.:20.12   1st Qu.:28.68   1st Qu.:12.77
 Median :21.16   Median :32.84   Median :13.66
 Mean   :20.97   Mean   :31.59   Mean   :13.42
 3rd Qu.:22.00   3rd Qu.:35.76   3rd Qu.:14.32
 Max.   :22.37   Max.   :37.66   Max.   :14.45

$trophic
hrbvr omnvr insct pscvr
    1     4    13     6
```

The `summary` function gives different types of results depending on the class of the object being summarised. For matrices such as `higgins$width`, `summary` returns various quantiles and means of the matrix columns, representing the three tributaries in this case. For other arrays such as `higgins$abundance`, `summary` gives these statistics over the entire array. For factors such as `higgins$trophic`, `summary` counts the number of observations in each category.

In short, `dlapply` attempts to repeatedly summarise data list margins by passing each variable to the standard `apply` function and then combining the successfully marginalised variables back into a data list. If this combination fails, `dlapply` will throw an error; in such a case, consider using the **base** R apply functions for lists (i.e., `lapply`; `sapply`; `rapply`) or the **plyr** package (i.e., `laply`; `ldply`; `llply`; `l_ply`).

4.3. Data list visualisation

Many standard plotting functions in the base R **graphics** package combine `formula` and `data.frame` objects to produce plots. Because `data.list` objects are readily coerced into data frames, these functions can also be used to visualise data lists. For example, the `boxplot` command can be used to visualise the variation in the abundances of each of the species across seasons and rivers (Figure 7),

```
R> data(higgins)
R> boxplot(
+   formula = sqrt(abundance) ~ species,
+   data = as.data.frame(dims_to_vars(higgins)),
+   horizontal = TRUE, las = 1, xaxt = 'n',
+   xlab = "abundance (square-root scale)", ylab = "species")
R> tickmarks <- with(higgins, pretty(sqrt(abundance)))
R> axis(1, at = tickmarks, labels = tickmarks^2)
```

Note the use of the `dims_to_vars` function, which is required because a dimension of replication (i.e., `species`) is referred to in the `formula` (see Section 2.7). Figure 7 shows that while many species are rare across all rivers and seasons (e.g., pyol; amna), some vary widely in abundance (e.g., dini; cyve).

One of the most challenging aspects of data with more than one dimension of replication is visualising this higher dimensional structure. For example, a single dimension of replication is naturally represented on a scatterplot as variation in the x-y position of the points. But other aesthetics beyond x-y position—such as colour or shape—are required to visualise multidimensional replication on a single scatterplot. We have found that the faceting (i.e., trellising) technique is an effective tool for visualising multiple dimensions of replication, when at least one of the dimensions is short (i.e., less than 30 replicates). Faceting is used to display subsets of the data in different panels, so that differences between the subsets can be more easily perceived (Wickham 2009). In the multiple-table context, dimensions of replication may be used to define these subsets.

To illustrate faceting in a multiple-table context, we ask whether environmental variables interact with species traits to affect abundance in the `higgins` data set. Visualising all of the possible trait-by-environment interactions is virtually impossible, and so we begin by considering a trait and environmental variable that have been hypothesised to interact in many stream fish communities: stream width and fish species life history (Goldstein and Meador 2004). Figure 8 provides a visualisation of the dependence of `abundance` (y-position) on both `width` (x-position) and `life.history` (point shape). Each panel is associated with a particular species—therefore, faceting represents the `species` dimension of replication. Within panels, each point represents a particular river in a particular season—therefore, the position of the points within each scatterplot represents both the `seasons` and `rivers` dimensions

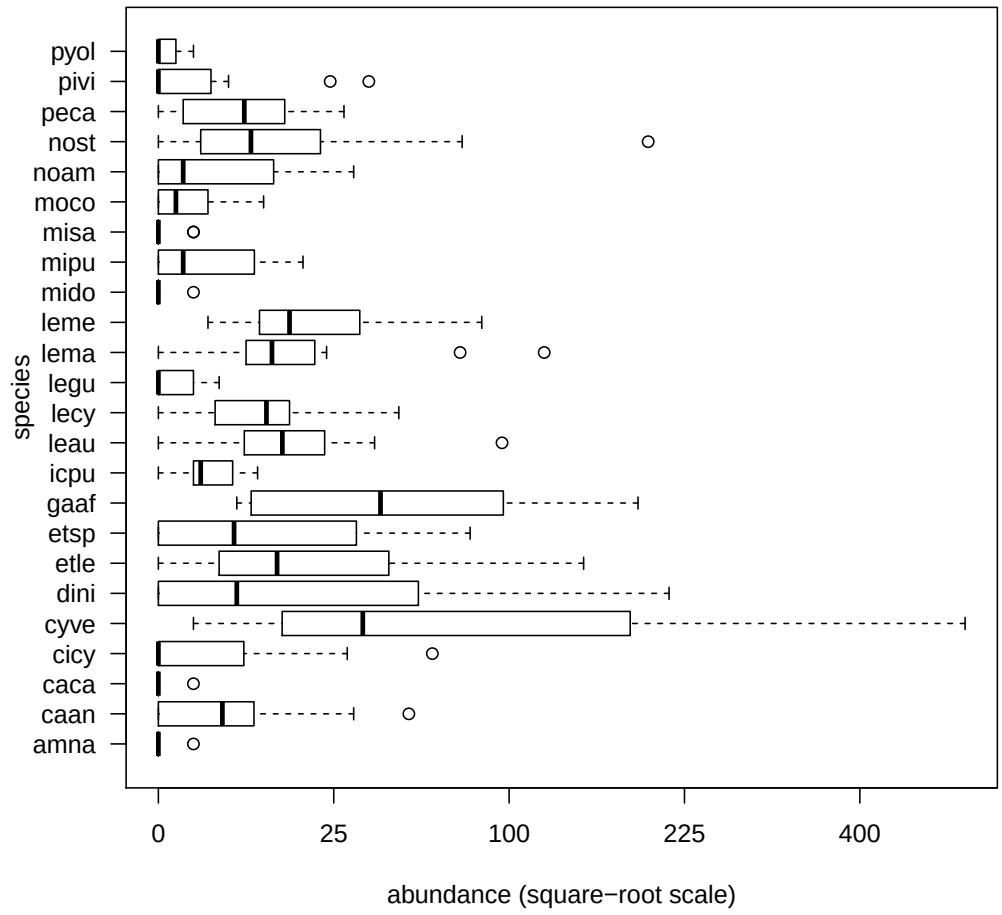


Figure 7: Boxplots showing spatio-temporal variation of the abundances of species in the `higgins` data.

of replication. Fitted model curves are also plotted for each species to help visualise any systematic trends in the scatterplots.

Figure 8 illustrates that the `width-abundance` relationship is variable along the `species` dimension of replication. However, this variation is not clearly related to the `life.history` trait that characterises `species`, although perhaps equilibrium-periodic species appear more likely than other species to have a flat width-abundance relationship. This possibility is investigated further in Sections 4.4-4.5 on statistical analysis.

The code for producing Figure 8 follows the same pattern as most applications of **multitable**: a data list is manipulated, it is converted to a data frame, and another R function is used to operate on the resulting data frame. Figure 8 requires two manipulations. Because we intend to use dimensions of replication to define the pattern of faceting, the `dimnames` of the data list must be accessible as variables; therefore, the `higgins` data must first be passed through the `dims_to_vars` function (Section 2.7),

```
R> data(higgins)
R> higgins <- dims_to_vars(higgins)
```

The order of the panels in Figure 8 is not arbitrary; the `species` dimension is ordered by `life.history`. This ordering helps to clarify whether certain types of species have similar `width-abundance` relationships, because the panels for similar species are adjacent in the faceted plot. To do this, the `species` variable must be converted into an ordered factor, with ordering determined by `life.history`; the `with` and `reorder` functions are useful for this purpose,

```
R> higgins$species <- with(higgins, reorder(species, life.history))
```

The `higgins` data are now ready to be coerced into a data frame. Before doing so, we load the **ggplot2** package,

```
R> library(ggplot2)
```

which we used to produce Figure 8. Because our focus is on describing **multitable**, we will not explain each **ggplot2** command in detail. However, the syntax of **ggplot2** functions should be readable to anyone familiar with R in general. One of the benefits of **ggplot2** is that the code for producing a plot can be broken into manageable pieces, which are added together to create the full plot (Wickham 2009). We begin by producing a `ggplot` object, `p`, for the `higgins` data,

```
R> p <- ggplot(as.data.frame(higgins))
```

In the next step we specify the pattern of faceting, which will represent replication along the `species` dimension,

```
R> p <- p + facet_wrap( ~ species, ncol = 4)
```

Here the `+` operator does not have its usual arithmetic meaning, but rather indicates that the property of faceting by species should be added to the plot object, `p`. Setting `ncol = 4` reflects a purely aesthetic choice to organise the scatterplots into four columns of six species each.

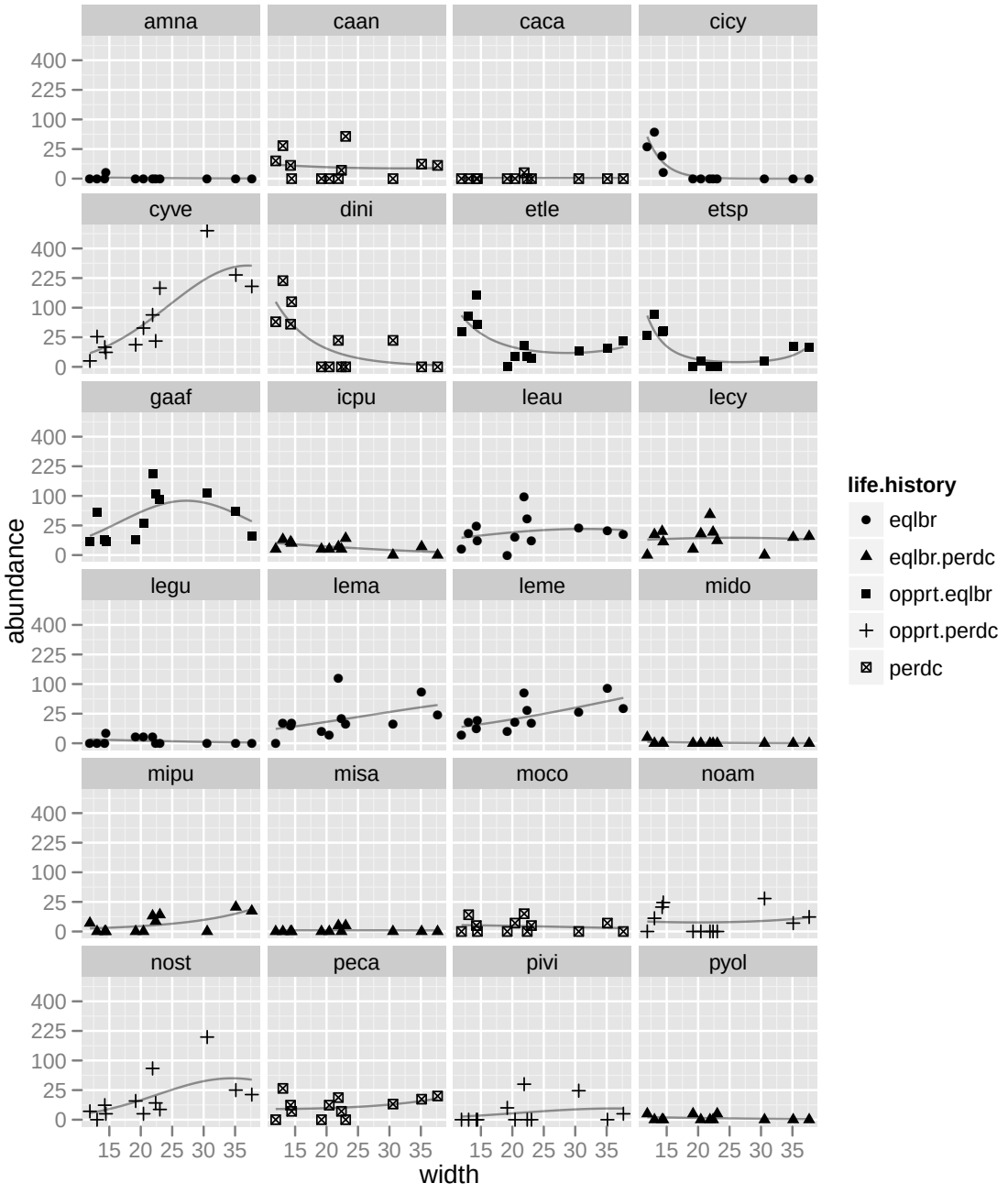


Figure 8: Visual exploration of the interaction between `width` and `life.history` on `abundance` in the `higgins` data. Faceting is used to represent the `species` dimension of replication.

Having specified that replication along the `species` dimension will be represented by faceting, we now specify that the remaining dimensions (`seasons` and `rivers`) will be represented by points,

```
R> p <- p + geom_point(aes(x = width, y = abundance, shape = life.history))
```

This command also indicates that the x-y position of the points will describe the `width` and `abundance` variables, whereas the shape of the points will describe `life.history`. To help guide the eye we plot a fitted quadratic predictive model of the `width-abundance` relationship in each panel using the `bayesglm` function from the **arm** package (Gelman, Su, Yajima, Hill, Pittau, Kerman, and Zheng 2011); the standard `glm` function in the **stats** package yielded unrealistically large coefficients, whereas `bayesglm` automatically shrinks coefficients towards zero resulting in a much better smoothing of the data.

```
R> library(arm)
R> p <- p + stat_smooth(aes(x = width, y = abundance), se = FALSE,
+     method = 'bayesglm', family = poisson, form = y ~ x + I(x^2),
+     colour = 'black', alpha = 0.4, geom = 'line')
```

Finally, the `abundance` data are put on a square-root scale, to help visually homogenise residual variance.

```
R> p <- p + scale_y_continuous(
+   trans = 'sqrt',
+   breaks = trans_breaks('sqrt', function(x) x^2))
```

And then the plot is produced (Figure 8),

```
R> print(p)
```

4.4. Generalised linear model example

Figure 8 suggested that `abundance` might have been affected by an interaction between `width` and `life.history`, because the abundances of species with an equilibrium-periodic life history had weaker relationships with `width` than other species. To explore this possibility we conducted a variety of formal statistical analyses, which also help to demonstrate how models can be fitted to data lists in much the same way that they are fitted to data frames.

For most model fitting functions in R, the hypothesised relationships between variables is specified via a `formula` object. For example, a simple formula including the effect of the interaction between `width` and `life.history` on `abundance` is given by,

```
R> form <- abundance ~ -1 + life.history + (scale(width):life.history)
```

The overall intercept has been removed (with `-1`) so that each `life.history` class has its own intercept. The `scale` command was used to standardise `width` to have mean zero and standard deviation one, which simplifies the interpretation of the estimated coefficients. We fitted this formula to the `higgins` data using the `glm` function with a `poisson` error distribution, which is a good first choice for count data. Because `glm` takes data in the form of a data frame, data lists can also be fitted by first passing them through `as.data.frame`,

```
R> higgins.glm <- glm(
+   form, family = poisson,
+   data = as.data.frame(higgins))
R> printCoefmat(summary(higgins.glm)$coefficients, signif.stars = FALSE)

                                    Estimate Std. Error  z value  Pr(>|z|)
life.historyeqlbr                   2.564697   0.033109  77.4613 < 2.2e-16
life.historyeqlbr.perdc             1.111861   0.067897  16.3756 < 2.2e-16
life.historyopprt.eqlbr             3.510256   0.029166 120.3560 < 2.2e-16
life.historyopprt.perdc             3.311290   0.030468 108.6820 < 2.2e-16
life.historyperdc                   2.125420   0.052921  40.1625 < 2.2e-16
life.historyeqlbr:scale(width)      0.234802   0.030539   7.6886 1.488e-14
life.historyeqlbr.perdc:scale(width)  0.137218   0.064942   2.1129   0.03461
life.historyopprt.eqlbr:scale(width) -0.218340   0.030926  -7.0600 1.665e-12
life.historyopprt.perdc:scale(width)  0.771661   0.022881  33.7250 < 2.2e-16
life.historyperdc:scale(width)      -0.935271   0.057963 -16.1357 < 2.2e-16
```

A striking aspect of this fitted model is that all of the coefficients are significant, and most are highly significant. However, these strong results are caused by inflated type I error rates related to the fact that repeated measurements have been taken on each species. Repeated measurements are *always* an important consideration when analysing data with multiple dimensions of replication; in a three-dimensional data list, such as ours for example, replication along two of the dimensions (e.g., `seasons` and `rivers`) induces repeated measurements of the replicates along the other (i.e., `species`). However, these inevitable repeated measurements will only inflate type I error if the deviations from model expectations tend to be more similar for observations associated with the same replicate along a particular dimension of replication—a form of autocorrelation. To visually explore this possibility, a good option is to plot the expected versus observed values of the response variable, faceted by a dimension of replication. For example, we plot `abundance` against its fitted values and facet by `species` (with 1:1 lines to guide the eye) (Figure 9),

```
R> higgins[['fitted', shape = 'abundance']] <-
+   array(fitted.values(higgins.glm), dim(higgins))
R> ggplot(as.data.frame(higgins)) +
+   facet_wrap( ~ species, ncol = 4) +
+   geom_point(aes(x = fitted, y = abundance)) +
+   geom_abline(intercept = 0, slope = 1) +
+   scale_y_continuous(trans = 'sqrt',
+     breaks = trans_breaks('sqrt', function(x) x^2)) +
+   scale_x_continuous(trans = 'sqrt',
+     breaks = trans_breaks('sqrt', function(x) x^2))
```

Autocorrelation is clearly visible in the panels for species within which all observed abundances lie below the 1:1 line (i.e., amna; legu; mido; misa; pyol; caca). The biological interpretation of this autocorrelation is that `life.history` does not completely determine each species' relationship with `width`, and therefore that `width` must interact with other traits to affect `abundance`.

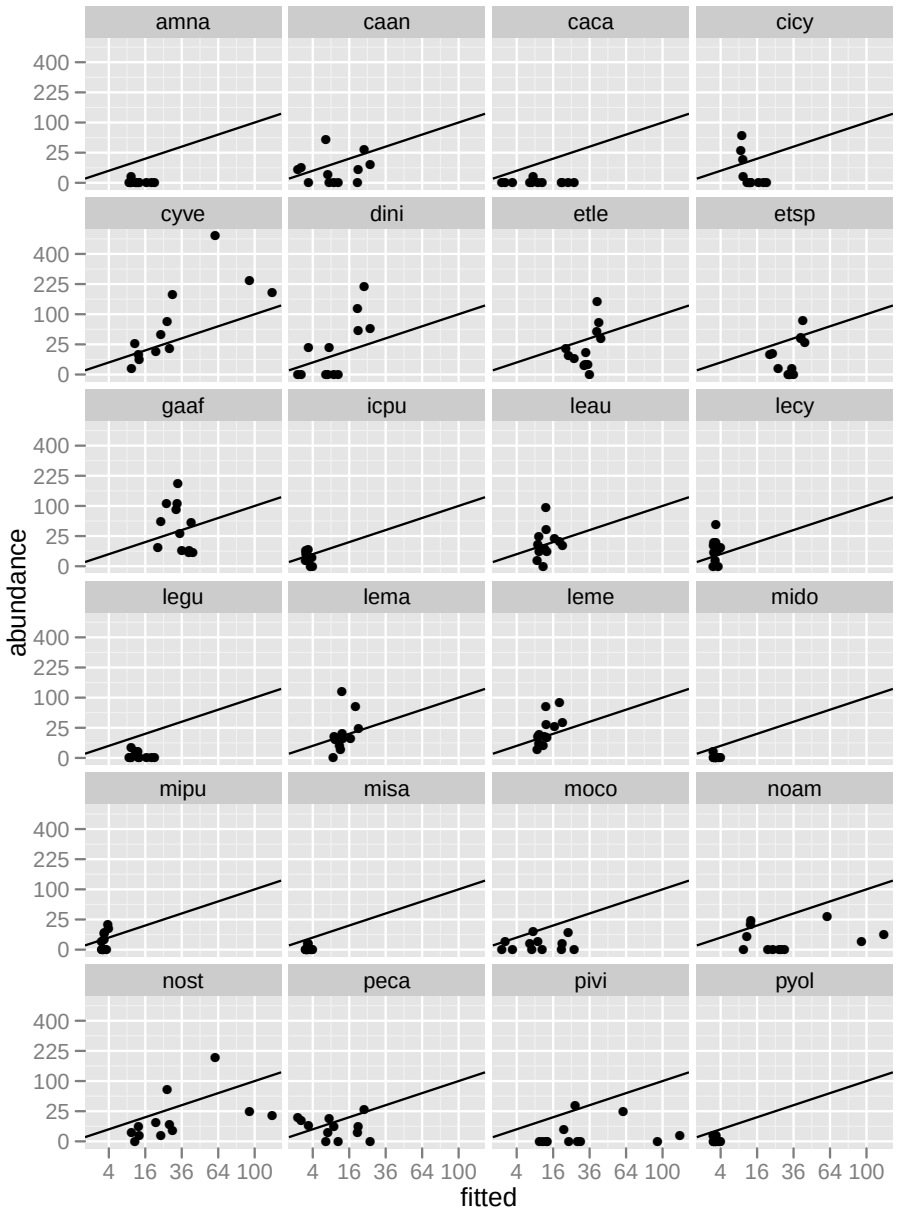


Figure 9: Observed abundances in the `higgins` data versus the abundances expected from a fitted generalised linear model with Poisson error structure and formula, `abundance ~ -1 + life.history + (scale(width):life.history)`. Each panel is for a single species, with 1:1 lines to aide visual interpretation.

Despite this autocorrelation issue, the parameter estimates themselves suggest that there may in fact be an interaction between `life.history` and `width`. In particular, the `width` coefficient for equilibrium-periodic species is smaller in magnitude than for the others. Furthermore, some `life.history` strategies were associated with positive `width` coefficients (i.e., equilibrium; equilibrium-periodic; and opportunistic-periodic) whereas the other two were associated with negative `width` coefficients (i.e., opportunistic-equilibrium and periodic). But, will this interaction be significant after properly accounting for autocorrelation? There are two general approaches to addressing this question: mixed modelling and randomisation testing, which is considered next.

4.5. Randomisation tests

As in all randomisation tests, the first step is to compute the statistics of the observed data, which in this case are the five coefficients defining the interaction between `life.history` and `width`.

```
R> coef.obs <- coefficients(higgins.glm)[6:10]
```

These observed coefficients will be compared with a distribution of coefficients for randomised data. Next we decide on the number of randomisations,

```
R> B <- 500
```

and allocate an array to store the randomised coefficients,

```
R> coef.B <- array(0, c(B, length(coef.obs), 2))
R> dimnames(coef.B) <- list(1:B, names(coef.obs), c('species','seasons.rivers'))
```

This array, `coef.B`, has one dimension for the `B` randomisations of the data, one for the number of coefficients in the model, and a third dimension for the number of null models to be considered; with more than a single dimension of replication, several null models are often required (e.g., Dray and Legendre 2008; Cormont, Vos, van Turnhout, Foppen, and ter Braak 2011). When sample sizes are sufficiently large, it is ideal to compute one null model for each dimension of replication. The null model for each dimension permutes the indices of the response variable (i.e., `abundance` in this case) for that dimension. This is the approach suggested by Dray and Legendre (2008). However, the `seasons` and `rivers` dimensions are very short (i.e., 4 and 3), which translates into only 24 and 6 possible permutations for these dimensions. Therefore, we consider only two null models: permute the (1) `species` dimension and (2) `seasons` and `rivers` dimensions.

Because randomisation tests involving data lists require the repeated coercion of data lists to data frames, we use the molding technique to speed up computation (Section 3.1). In this example, the savings in speed are modest, but they can be much greater in larger problems. Molding begins by taking a mold of the initial data list,

```
R> mold.higgins <- data.list.mold(higgins)
```

Both null models require a three-line loop over the `B` permutations,

```
R> higgins.tmp <- higgins
R> for(i in 1:B){
+   higgins.tmp$abundance <- higgins$abundance[sample(24),,]
+   df.tmp <- as.data.frame(higgins.tmp, mold = mold.higgins)
+   coef.B[i, , 1] <- glm(form, family = poisson, df.tmp)$coefficients[6:10]
+ }
```

The first line permutes the `species` dimension of the response variable; the second coerces the permuted data list to a data frame; and the third computes and stores the coefficients estimated from the permuted data. The second null model is computed similarly,

```
R> higgins.tmp <- higgins
R> for(i in 1:B){
+   higgins.tmp$abundance <- higgins$abundance[,sample(4),sample(3)]
+   df.tmp <- as.data.frame(higgins.tmp, mold = mold.higgins)
+   coef.B[i, , 2] <- glm(form, family = poisson, df.tmp)$coefficients[6:10]
+ }
```

To avoid issues with two-sided tests, we assess the magnitudes of the coefficients irrespective of their signs,

```
R> coef.B.abs <- abs(coef.B)
R> coef.obs.abs <- abs(coef.obs)
```

Simple estimates of p-values can now be computed by processing `coef.B.abs` using standard R commands,

```
R> pvalues <- apply(sweep(coef.B.abs, 2, coef.obs.abs, '>='), c(2,3), mean)
R> round(cbind(pvalues, coef.obs), 3)
```

```
                                    species seasons.rivers coef.obs
life.historyeqlbr:scale(width)        0.626          0.336    0.235
life.historyeqlbr.perdc:scale(width)  0.780          0.532    0.137
life.historyopprt.eqlbr:scale(width)  0.708          0.274   -0.218
life.historyopprt.perdc:scale(width)  0.122          0.474    0.772
life.historyperdc:scale(width)        0.072          0.192   -0.935
```

With these more accurate p-values, we can see that there is very little evidence for an interaction between `width` and `life.history`. A mixed-model approach that more directly accounts for repeated measurements on species might be beneficial here. Although such an approach is beyond our scope here, the consistent logic of `multitable` applies: manipulate the data list, coerce it to a data frame, and pass the data frame to a mixed-model fitting function.

4.6. Analysing data lists with multivariate methods

So far we have used data lists by manipulating them, converting to a data frame, and passing to a function to produce an analysis or visualisation. In the context of multivariate analysis,

it is often useful to cut out the middle step and pass a data list directly to a function. In most univariate methods, both the response and explanatory variables are vectors; in contrast, the response variables of many multivariate methods are matrices while the explanatory variables are vectors. Because data lists are—unlike data frames—able to store both matrix- and vector-valued variables, they are well suited for use in multivariate analyses.

As an example of using a data list in a multivariate analysis, we use the `adonis` (analysis of dissimilarities) function in the **vegan** package (Oksanen, Blanchet, Kindt, Legendre, Minchin, O'Hara, Simpson, Solymos, Stevens, and Wagner 2011) to analyse the relationships between `abundace` and the environmental variables in the `higgins` data. However, `abundance` is a three dimensional array, and `adonis`—like many other multivariate methods requires a matrix-valued response variable. Therefore, `higgins` must be manipulated before it is ready to be analysed. In particular, we collapse the `seasons` and `rivers` dimensions of replication into a single dimension, `seasons.rivers`. In Section 2.8 we described how to collapse dimensions using the melt-recast concept, and so we use this technique here,

```
R> data(higgins)
R> higgins.melt <- dlmelt(higgins)
R> higgins.melt$species.seasons.rivers <-
+   within(higgins.melt$species.seasons.rivers, {
+     seasons.rivers <- interaction(seasons, rivers)
+     seasons <- rivers <- NULL
+ })
R> higgins.melt$seasons.rivers <-
+   within(higgins.melt$seasons.rivers,
+     seasons.rivers <- interaction(seasons, rivers)
+ )
R> higgins <- dlcast(higgins.melt)
```

This new version of `higgins` has only two dimensions of replication, with `abundance` as a matrix and the environmental variables as one-dimensional vectors,

```
R> summary(higgins)
```

```
               abundance width  temp depth velocity substrate habitat seasons
species             TRUE FALSE FALSE FALSE    FALSE     FALSE   FALSE   FALSE
seasons.rivers      TRUE  TRUE  TRUE  TRUE     TRUE      TRUE    TRUE    TRUE
               rivers trophic life.history
species         FALSE    TRUE         TRUE
seasons.rivers   TRUE   FALSE        FALSE
```

We can now call adonis within an environment defined by `higgins`,

```
R> library(vegan)
R> dl.adonis <- with(higgins, adonis(
+   t(abundance) ~ width + temp + depth + velocity + substrate + habitat,
+   strata = dl$species
+ ))
R> print(dl.adonis$aov.tab, signif.stars = FALSE)
```

```
           Df SumsOfSqs MeanSqs F.Model      R2 Pr(>F)
width       1   0.80370 0.80370  6.3398 0.28374  0.001
temp        1   0.28776 0.28776  2.2699 0.10159  0.046
depth       1   0.19041 0.19041  1.5020 0.06722  0.193
velocity    1   0.43200 0.43200  3.4078 0.15252  0.011
substrate   1   0.23012 0.23012  1.8152 0.08124  0.104
habitat     1   0.25466 0.25466  2.0088 0.08991  0.082
Residuals   5   0.63385 0.12677         0.22378
Total      11   2.83250                 1.00000
```

The fact that `width` and `velocity` have significant effects on `abundance` indicates that these factors might be important in structuring these stream fish communities. However, taken together with the previous results using Poisson generalised linear models, these results also indicate that these effects are not well explained by the `life.history` trait.

5. Conclusion

The structure of `data.list` objects is sufficiently rich to give rise to a wider variety of uses than can be described in detail here. Our intention was to illustrate the basic features and concepts of the **multitable** package, and to demonstrate its utility. Our long-term goal with the **multitable** project in general is to make standard analyses in R simpler to conduct on complex multiple-table data.

Acknowledgements

We thank Laura Timms and two anonymous reviewers for helpful comments on an earlier version of the manuscript; Levi Waldron, Ben Bolker, and Philip Dixon for discussions and suggestions about software design; and Beatrix Beisner for discussions about biology. We also thank Higgins (2009) for making his data freely available.

References

Cantin A, Beisner BE, Gunn JM, Prairie YT, Winter JG (2011). "Effects of Thermocline Deepening on Lake Plankton Communities." *Canadian Journal of Fisheries and Aquatic Science*, **68**, 260–276.

Chambers JM, Hastie TJ (1992). *Statistical Models in S*. Wadsworth and Brooks, Pacific Grove, California.

Cormont A, Vos CC, van Turnhout CA, Foppen RP, ter Braak CJ (2011). "Using life-history traits to explain bird population responses to changing weather variability." *Climate Research*, **49**, 59–71.

Dolédec S, Chessel D, ter Braak C, Champely S (1996). "Matching Species Traits to Environmental Variables: A New Three-Table Ordination Method." *Environmental and Ecological Statistics*, **3**, 143–166.

Dray S, Legendre P (2008). "Testing the Species Traits-Environment Relationships: The Fourth-Corner Problem Revisited." *Ecology*, **89**(12), 3400–3412.

Gelman A, Su YS, Yajima M, Hill J, Pittau MG, Kerman J, Zheng T (2011). *arm: Data analysis using regression and multilevel/hierarchical models*. R package 1.4-14, URL http://CRAN.R-project.org/package=arm.

Goldstein RM, Meador MR (2004). "Comparisons of Fish Species Traits from Small Streams to Large Rivers." *Transactions of the American Fisheries Society*, **133**, 971–983.

Higgins CL (2009). "Spatiotemporal variation in functional and taxonomic organization of stream-fish assemblages in central Texas." *Aquatic Ecology*, **43**, 1133–1141.

Ives AR, Helmus MR (2011). "Generalized Linear Mixed Models for Phylogenetic Analyses of Community Structure." *Ecological Monographs*, **81**(3), 511–523.

Legendre P, Galzin R, Harmelin-Vivien ML (1997). "Relating Behavior to Habitat: Solutions to the Fourth-Corner Problem." *Ecology*, **78**(2), 547–562.

Leibold MA, Economo EP, Peres-Neto PR (2010). "Metacommunity Phylogenetics: Separating the Roles of Environmental Filters and Historical Biogeography." *Ecology Letters*, **13**, 1290–1299.

Oksanen J, Blanchet FG, Kindt R, Legendre P, Minchin PR, O'Hara R, Simpson GL, Solymos P, Stevens MHH, Wagner H (2011). *vegan: Community ecology package*. R package version 2.0-2, URL http://CRAN.R-project.org/package-vegan.

Pillar VD, Duarte LD (2010). "A Framework for Metacommunity Analysis of Phylogenetic Structure." *Ecology Letters*, **13**, 587–596.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sólymos P (2009). "Processing Ecological Data in R with the **mefa** Package." *Journal of Statistical Software*, **29**(8), 1–28.

Wickham H (2007). "Reshaping Data with the **reshape** Package." *Journal of Statistical Software*, **21**(12).

Wickham H (2009). *ggplot2: elegant graphics for data analysis*. Springer New York.

Wickham H (2011). "The split-apply-combine strategy for data analysis." *Journal of Statistical Software*, **40**(1), 1–29.

Affiliation:

Steven C Walker
Département de Sciences Biologiques
Université de Montréal

C.P.6128, Succursale Centre-ville
Montréal, Québec, H3C 3J7 Canada
E-mail: steve.walker@utoronto.ca
URL: http://sites.google.com/site/stevencarlislewalker/