Multiple-table data in R

SC Walker

September 20, 2011

Abstract

Data frames are integral to R. They provide a standard format for passing data to model-fitting and plotting functions. This standard makes it easier for experienced useRs to learn new functions, because most developments in R continue to accept a single data frame as input. Still, many data sets do not easily fit into a single data frame, and my field of community ecology provides many examples of such inherently multiple-table data (e.g. fourth-corner problem and other trait-based data sets). Storing such data in a single data frame results in either large numbers of meaningless missing values or storage of redundant information. These storage problems have led ecologists to model summaries of their data (e.g. community-weighted trait matrices), rather then their data itself. Perhaps more importantly, my experience with manipulating such data using data frames has resulted in difficult-to-read workflows with many lines. The `multitable` package introduces new data storage objects called `data.list`s, which are extensions of `data.frame`s. As `data.list`s can be coerced to `data.frame`s, they can be used with all R functions that accept an object that is coercible to a `data.frame` (e.g. `lm`; `plot`; `lme`; and many more). The `multitable` package also provides several mechanisms for simplifying the manipulation of `data.list` objects.

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 and variables respectively. Standard R workflows require that all of the data to be analyzed are organized into a single data frame, and 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) (Chambers and Hastie, 1992). This framework allows scientists to concentrate on their primary interests—the relationships between variables—without explicit reference to complex mathematical and algorithmic details. It also provides access to those details, which are required (1) for more effective analyses and (2) 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. Thus, by separating low-level methods development from high-level data analysis, R fosters the formation of a community of researchers where both methodologists and analysts can have mutually beneficial interactions.

However, research in my field of community ecology has led my colleagues and I to data sets that do not easily fit within a single data frame. A common example is the fourth-corner problem (Legendre et al., 1997), in which three data tables are to be analyzed: 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 (Fig. 1). Such data are characterized by a conspicuous (lower-right) 'fourth-corner', where there are no data. These fourth-corners of missing data 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, Fig. 2). The challenge of analyzing such multiple-table data sets in R is that it is not obvious how to organize them into a single `data.frame`, which is required in standard R workflows. Our goal with the `multitable` package is to provide tools for analyzing multiple-table data sets within this standard R framework.

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 has been a fair amount of work in this direction, focusing on data with a fourth-corner problem (Dolédec et al., 1996; Legendre et al., 1997; Dray and Legendre, 2008; Pillar and Duarte, 2010; Leibold et al., 2010; Ives and Helmus, 2011). However, this work does not apply to data sets that have other more complex multiple-table data structures (e.g. zooplankton communities in Lac Croche, Fig. 2; Ref ??). One approach to such issues would be to build new data analysis functions for each new data structure. But such an approach is less than ideal, as it would require that new methods be learned for each new structure—it does not take advantage of the large number of tools developed within the standard R framework of data frames and formulas. The `multitable` package provides an alternative approach, by introducing a multiple-table generalization of data frames—called data lists—which can be analyzed with virtually any function that can be used to analyze a data frame. Thus, instead of providing new methods of analysis, `multitable` provides new methods of data management.

There are several existing R packages that are designed to make data management easier (e.g. `reshape2`; Wickham, 2007). In fact, the `mefa` and `mefa4` packages have been developed to organize data with a slight generalization[1] of the fourth-corner problem (Sólymos, 2009). The `multitable` package has much in common with `mefa`, but there are noticeable differences. For example, `mefa`

[1]Several community matrices—called segments—with identical dimensions are allowed in `mefa`.

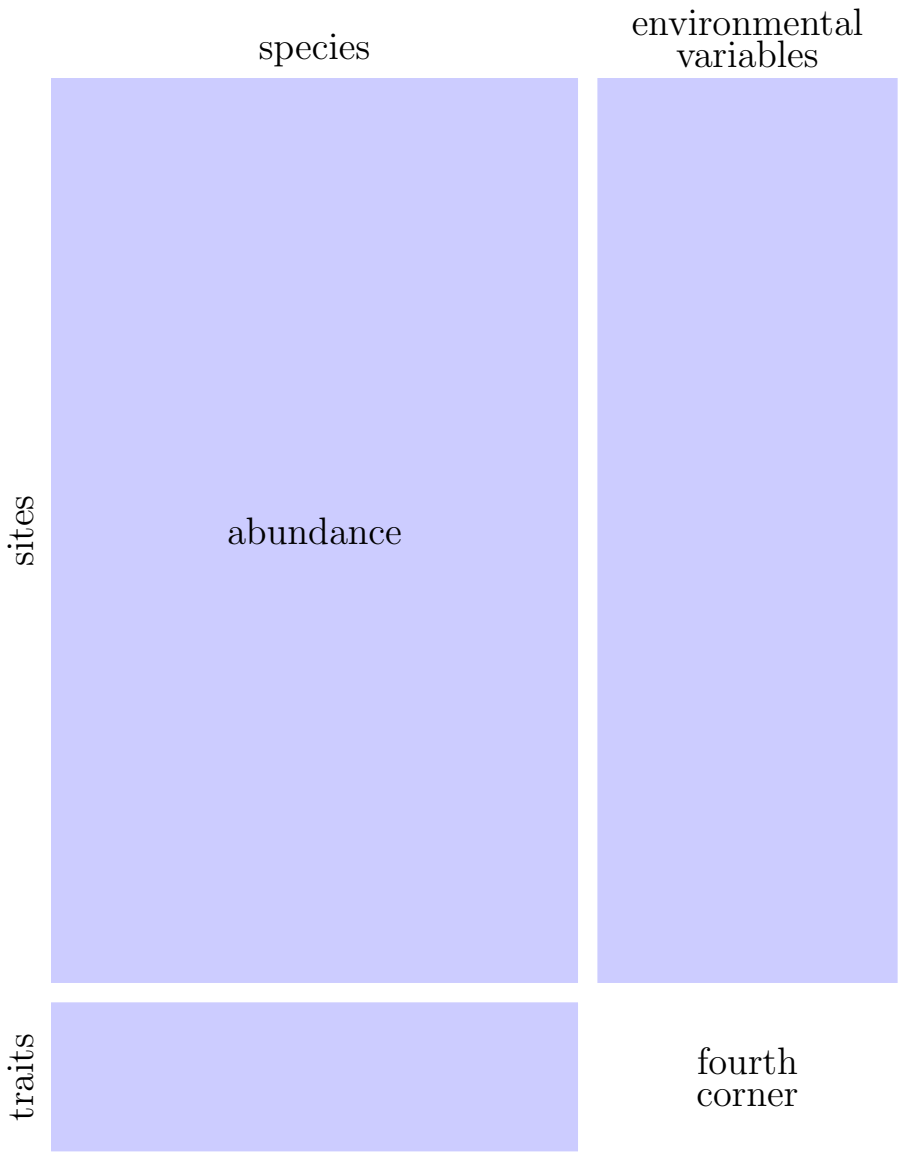

Figure 1: Fourth corner problem.

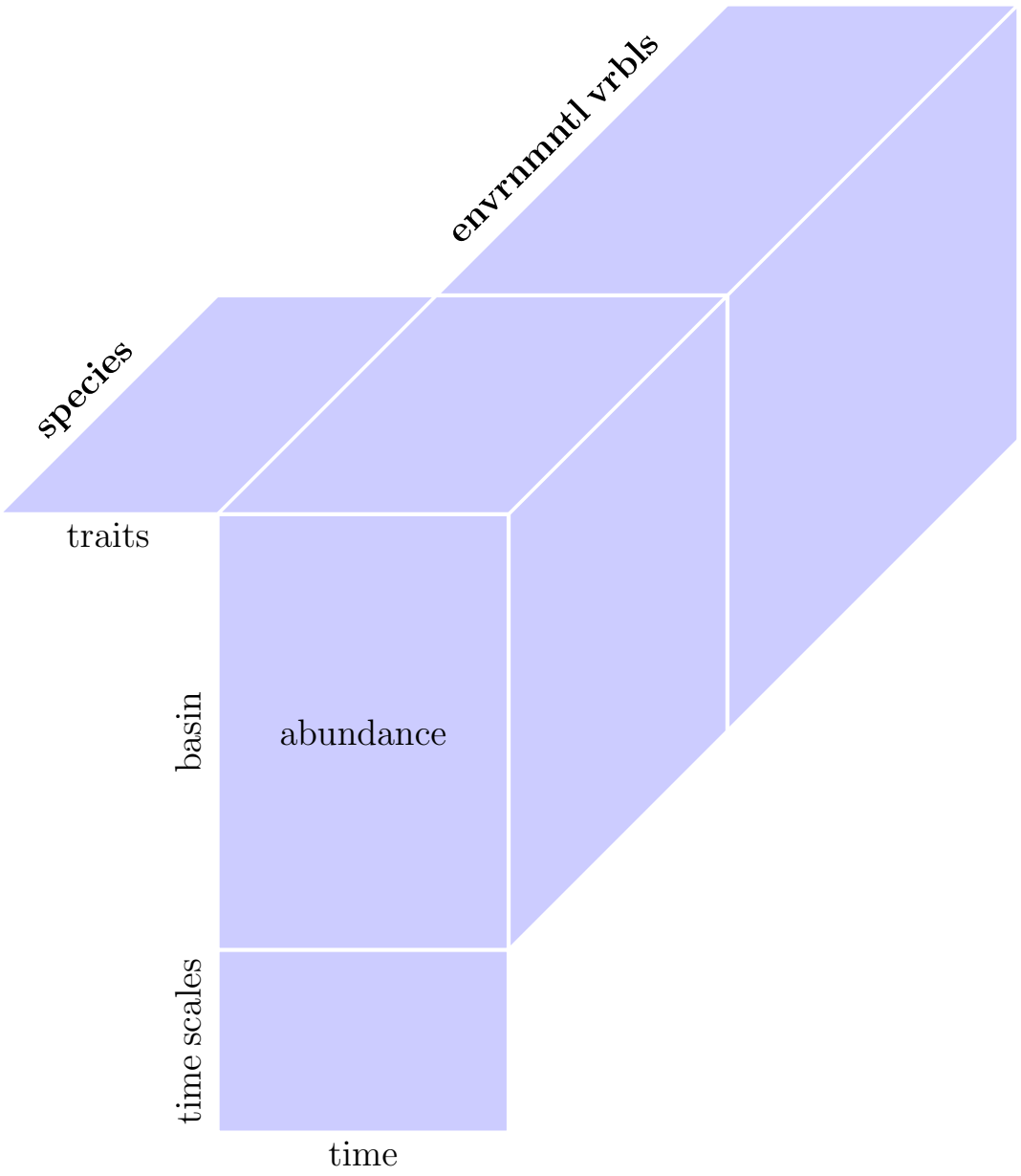

Figure 2: The structure of the Lac Croche zooplankton community data.

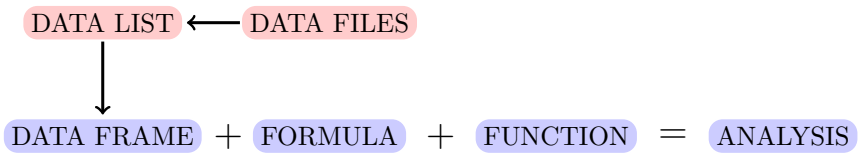

Figure 3: The `multitable` paradigm for including multiple-table data (in red) into the standard R workflow (in blue). Data lists are used to organize and manipulate multiple-table data as a single R object, even though it must be stored in multiple text-based data files. When such data are required for analysis, they are coerced into a data frame. Once in data frame form, they can be used in analyses by combining them with formulas (to specify hypothetical relationships between variables) and functions (to call computational methods).

provides more extensive tools for data summarization than `multitable` and `mefa4` integrates tools for sparse-matrix computations. On the other hand, `multitable` is designed to handle more general data structures than `mefa` or `mefa4` (e.g. `mefa` cannot organize the Lac Croche data structure, Fig. 2). However, we hope that `mefa` and `multitable` will be complementary, not competitive.

The `multitable` model of data organization is illustrated in Figure 3 (`mefa` uses a similar model). The elements of the standard R workflow are in blue: data frames; formulas; functions; and analyses. The `multitable` package seeks to facilitate the use of such workflows with multiple-table data by creating tools (arrows and red boxes) for organizing and manipulating such data. These tools are based on a new kind of object, called a data list, which is used to organize multiple-table data. Data lists can be manipulated much like data frames (e.g. variables can be transformed; groups of observations extracted or removed). Our design principle was to keep the manipulation of data lists as similar as possible to the manipulation of data frames. Once data lists are ready for analysis `multitable` provides tools for coercing them into data frames, thereby entering the standard R workflow. Importantly, data, formulas, and functions are kept separate, thus preserving the benefits of using R in this standard way.

The purpose of this vignette is to justify and introduce the use of the `multitable` package. I begin by describing the structure of an example `data.list` object. Then I illustrate one of the most powerful features of `data.lists`: methods that allow related variables, which cannot fit into a single data frame, to be subscripted simultaneously. Next I show that variables in data lists can be transformed and modeled, in the same manner that is standard for variables in data frames. Finally, I describe a simple method for creating a data list of your own data, and use this method to introduce some useful concepts associated with multiple-table data.

2 The structure of data lists

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

```
> library(multitable)
> data(fake.community)
> 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 in 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 a `dim` (i.e. dimension) attribute, which gives it 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 indicated above after the variable itself is printed. Some variables are only replicated along two dimensions (e.g. temperature and precipitation) and others only a single 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.

```
> 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 table 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 bio-physical necessities, whereas some 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 physically and logically impossible to measure the precipitation of a species, and so this `FALSE` is necessarily `FALSE`.

3 Subscripting data lists

This structure relating variables and dimensions of replication, allows us to manipulate multiple variables simultaneously. In particular, `multitable` makes it possible to extract pieces of a data list while maintaining its structure. 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.

```
> 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 on the first species (i.e. capybara) in 1537 for the first three sites.

```
> fake.community[1:3,"1537",1]
abundance:
---------
, , capybara

            1537
midlatitude    0
subtropical    0
tropical       0

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


temperature:
-----------
            1537
midlatitude   NA
subtropical   NA
tropical      NA
Replicated along:  || sites || years ||


precipitation:
-------------
            1537
midlatitude   NA
subtropical   NA
tropical      NA
Replicated along:  || sites || years ||


body.size:
```

```
---------
capybara
     140
Replicated along:  || species ||


metabolic.rate:
--------------
capybara
      20
Replicated along:  || species ||


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


REPLICATION DIMENSIONS:
  sites   years species
      3       1       1
```

Notice also that for each different subset of the data, the new replication dimensions are printed after the data.

4 Transforming variables in data lists

Often we need to transform variables before passing data frames to functions. This is easily done with variables in data lists as well. For example, suppose we want to make a $\log(x + 1)$ transformation of the abundance data.

```
> fake.community$abundance <- log(fake.community$abundance + 1)
> fake.community$abundance
, , capybara

               2009     2008 1537
midlatitude 1.609438 0.000000    0
subtropical 0.000000 2.397895    0
tropical    2.197225 0.000000    0
equatorial  0.000000 2.079442    0
arctic      0.000000 0.000000    0
subarctic   0.000000 0.000000    0
```

```
, , moss

               2009     2008 1537
midlatitude 0.000000 1.945910    0
subtropical 0.000000 0.000000    0
tropical    2.302585 0.000000    0
equatorial  0.000000 1.386294    0
arctic      1.791759 0.000000    0
subarctic   0.000000 0.000000    0

, , vampire

            2009 2008      1537
midlatitude    0    0 0.0000000
subtropical    0    0 0.6931472
tropical       0    0 0.0000000
equatorial     0    0 0.0000000
arctic         0    0 0.0000000
subarctic      0    0 0.0000000

attr(,"subsetdim")
  sites   years species
   TRUE    TRUE    TRUE
```

We note that `fake.community` has a lot of missing values, which were useful for illustrating how data lists handle missing values, but will make further illustrations somewhat underwhelming. We can replace these missing values with values using the standard logic of R replacement.

```
> fake.community$temperature[,"1537"] <-
         c(5,10,30,20,-80,-10)
> fake.community$precipitation[,"1537"] <-
         c(5,50,75,50,2,7)
> fake.community$body.size["moss"] <- 1
```

5 Simple analysis functions

Data lists can be passed 'as is' to many standard functions in R that normally take data frames. In the next section I will define this class of functions in more detail, but for now consider this simple example. Perhaps we want to explore whether the interaction between body size and temperature has an influence on abundance. As a first attempt at model building, we fit a linear model using `lm`.

```
> lm(abundance ~ (body.size*temperature),data=fake.community)
```

```
Call:
lm(formula = abundance ~ (body.size * temperature), data = fake.community)

Coefficients:
          (Intercept)              body.size
            4.484e-01             -1.718e-03
          temperature  body.size:temperature
            3.634e-03              5.041e-07
```

And this works just as well with mixtures of categorical and numerical data.

```
> lm(abundance ~ -1+(homeotherm*temperature),data=fake.community)
Call:
lm(formula = abundance ~ -1 + (homeotherm * temperature), data = fake.community)

Coefficients:
            homeothermN              homeothermY
               0.228770                 0.318948
            temperature  homeothermY:temperature
               0.001186                 0.007512
```

It also works with other 'simple' functions, such as `rlm` (robust linear model) in the `MASS` package.

```
> library(MASS)
> rlm(abundance ~ (body.size*temperature),data=fake.community)
Call:
rlm(formula = abundance ~ (body.size * temperature), data = fake.community)
Converged in 10 iterations

Coefficients:
          (Intercept)              body.size
         2.606699e-05         -1.076827e-07
          temperature body.size:temperature
         3.043212e-07         -8.994997e-10

Degrees of freedom: 51 total; 47 residual
  (3 observations deleted due to missingness)
Scale estimate: 5.26e-05
```

Therefore, in many cases, data lists enter the standard R workflow in exactly the same manner as data frames.

6 Coercing data lists to data frames

The reason that unmodified data lists can be passed to some functions that are expecting data frames, is that these functions try to coerce whatever data object

they receive into a data frame. When the `multitable` package is loaded, these functions can find a method for making such a conversion. This method can be accessed by users directly via the `as.data.frame` function from the R `base` package. For example, we can pass the `fake.community` data to `as.data.frame`.

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

	abundance	temperature	precipitation	body.size	metabolic.rate	h
midlatitude.2009.capybara	1.6094379	NA	NA	140	20	
subtropical.2009.capybara	0.0000000	25	99	140	20	
tropical.2009.capybara	2.1972246	48	149	140	20	
equatorial.2009.capybara	0.0000000	50	199	140	20	
arctic.2009.capybara	0.0000000	-37	21	140	20	
subarctic.2009.capybara	0.0000000	3	41	140	20	
midlatitude.2008.capybara	0.0000000	10	20	140	20	
subtropical.2008.capybara	2.3978953	20	100	140	20	
tropical.2008.capybara	0.0000000	50	150	140	20	
equatorial.2008.capybara	2.0794415	30	200	140	20	
arctic.2008.capybara	0.0000000	-30	20	140	20	
subarctic.2008.capybara	0.0000000	0	40	140	20	
midlatitude.1537.capybara	0.0000000	5	5	140	20	
subtropical.1537.capybara	0.0000000	10	50	140	20	
tropical.1537.capybara	0.0000000	30	75	140	20	
equatorial.1537.capybara	0.0000000	20	50	140	20	
arctic.1537.capybara	0.0000000	-80	2	140	20	
subarctic.1537.capybara	0.0000000	-10	7	140	20	
midlatitude.2009.moss	0.0000000	NA	NA	1	5	
subtropical.2009.moss	0.0000000	25	99	1	5	
tropical.2009.moss	2.3025851	48	149	1	5	
equatorial.2009.moss	0.0000000	50	199	1	5	
arctic.2009.moss	1.7917595	-37	21	1	5	
subarctic.2009.moss	0.0000000	3	41	1	5	
midlatitude.2008.moss	1.9459101	10	20	1	5	
subtropical.2008.moss	0.0000000	20	100	1	5	
tropical.2008.moss	0.0000000	50	150	1	5	
equatorial.2008.moss	1.3862944	30	200	1	5	
arctic.2008.moss	0.0000000	-30	20	1	5	
subarctic.2008.moss	0.0000000	0	40	1	5	
midlatitude.1537.moss	0.0000000	5	5	1	5	
subtropical.1537.moss	0.0000000	10	50	1	5	
tropical.1537.moss	0.0000000	30	75	1	5	
equatorial.1537.moss	0.0000000	20	50	1	5	
arctic.1537.moss	0.0000000	-80	2	1	5	
subarctic.1537.moss	0.0000000	-10	7	1	5	
midlatitude.2009.vampire	0.0000000	NA	NA	190	0	

```
subtropical.2009.vampire  0.0000000          25           99       190              0
tropical.2009.vampire     0.0000000          48          149       190              0
equatorial.2009.vampire   0.0000000          50          199       190              0
arctic.2009.vampire       0.0000000         -37           21       190              0
subarctic.2009.vampire    0.0000000           3           41       190              0
midlatitude.2008.vampire  0.0000000          10           20       190              0
subtropical.2008.vampire  0.0000000          20          100       190              0
tropical.2008.vampire     0.0000000          50          150       190              0
equatorial.2008.vampire   0.0000000          30          200       190              0
arctic.2008.vampire       0.0000000         -30           20       190              0
subarctic.2008.vampire    0.0000000           0           40       190              0
midlatitude.1537.vampire  0.0000000           5            5       190              0
subtropical.1537.vampire  0.6931472          10           50       190              0
tropical.1537.vampire     0.0000000          30           75       190              0
equatorial.1537.vampire   0.0000000          20           50       190              0
arctic.1537.vampire       0.0000000         -80            2       190              0
subarctic.1537.vampire    0.0000000         -10            7       190              0
```

The resulting data frame contains one column for each variable and one row for each combination of replicates across the three dimensions of replication. 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 the traits are only replicated along species there are only three unique trait 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.

By storing these data in a single data frame, we can now pass them to any function that accepts data frames. For example, we can graphically examine the interaction between an environmental variable and a trait using the `xyplot` function from the `lattice` package (Fig. 4). Because these are completely fake data I won't make too much out of the results, but there doesn't seem to be much of an interaction between body size and temperature. The exciting thing about this graph is that it suggests a general graphical method for exploring the interactions between traits and environmental variables on community composition.

On occasion, one may wish to iteratively coerce a sequence of data lists to data frames. For example, in a randomization test one might loop over a number of random subscripts of a data list. In such a case, each resulting data frame has the same structure (i.e. the same replication dimensions and variables).

```
> fake.community.rep <- fake.community
> b <- rep(0,100)
> for(i in 1: 100){
         fake.community.rep$abundance  <- fake.community$abundance[sample(6),,]
         rep.df <- as.data.frame(fake.community.rep)
         b[i] <- lm(abundance ~ temperature * body.size,
```

```
> library(lattice)
> xyplot(abundance ~ temperature | body.size,data=fake.community.df)
```

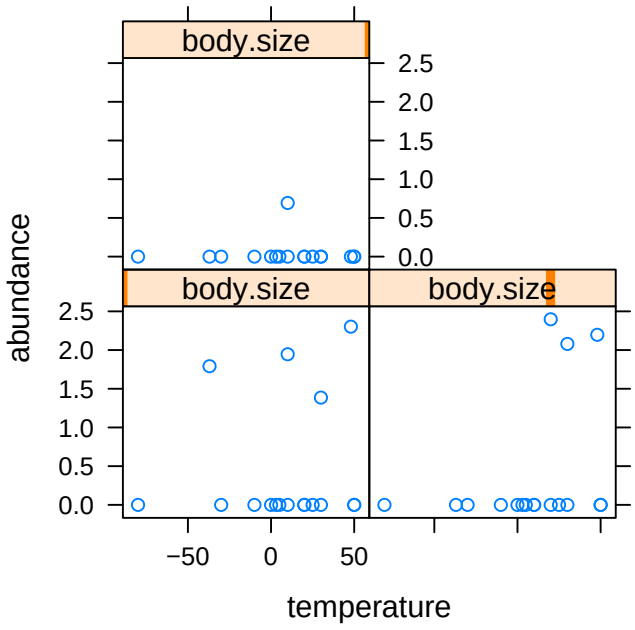

Figure 4: `xyplot`

```
                rep.df)$coefficient[4]
 }
> b
  [1] -1.794674e-05 -5.051363e-05  3.256359e-05
  [4] -1.350768e-05 -1.122857e-05 -4.314527e-05
  [7] -1.298889e-05 -1.298889e-05 -1.122857e-05
 [10] -1.254093e-05 -3.660504e-05 -6.759472e-05
 [13]  2.186549e-05 -3.017058e-05  2.961768e-05
 [16] -1.597213e-05 -1.615054e-05  2.775605e-05
 [19] -1.298347e-05  9.873051e-06  5.062938e-06
 [22] -3.898240e-05 -1.285979e-05 -6.524709e-07
 [25]  3.006222e-06  9.605852e-06 -3.455414e-05
 [28] -4.953546e-06  9.469401e-06  5.565095e-06
 [31]  7.657613e-06  2.961768e-05 -1.844299e-05
 [34]  3.176205e-05 -7.859302e-06 -3.660504e-05
 [37]  6.660457e-06 -3.173844e-06 -3.455414e-05
 [40]  1.496169e-05 -2.128748e-05  2.971648e-05
 [43]  4.235577e-06 -1.719530e-06 -1.754733e-06
 [46]  1.767926e-05 -2.761855e-05 -1.110886e-05
 [49] -7.133141e-07  2.314871e-05 -1.899313e-05
 [52] -2.316709e-05 -2.128748e-05 -4.471830e-05
 [55] -1.660765e-05  5.565095e-06  2.398663e-06
 [58] -2.316709e-05  1.436972e-06 -1.254420e-05
 [61]  2.971648e-05 -7.059237e-05 -6.904866e-06
 [64] -1.590195e-05 -6.519590e-05 -4.596965e-05
 [67] -3.017058e-05 -2.211790e-06  1.496169e-05
 [70] -1.622006e-05 -4.204165e-05 -4.989154e-05
 [73] -5.173717e-05 -8.757638e-06 -5.108739e-05
 [76] -4.743254e-05 -1.254093e-05 -1.548618e-05
 [79] -5.351039e-05  1.801282e-05  5.525808e-06
 [82] -4.583611e-05  9.519368e-06  1.498493e-05
 [85] -3.022373e-05  1.453285e-05 -2.735201e-05
 [88] -4.456175e-06 -4.367071e-05  1.724732e-05
 [91] -3.416494e-06 -1.683127e-05  4.601231e-06
 [94] -1.972856e-05 -3.358484e-06  1.995447e-05
 [97] -4.964689e-05 -2.923433e-05 -5.409303e-05
[100] -4.727717e-05


> fake.community.rep <- fake.community
> mold <- data.list.mold(fake.community)
> b <- rep(0,100)
> for(i in 1: 100){
         fake.community.rep$abundance  <- fake.community$abundance[sample(6),,]
         rep.df <- as.data.frame(fake.community.rep,
                 mold = mold)
```

```
        b[i] <- lm(abundance ~ temperature * body.size,
                rep.df)$coefficient[4]
 }
> b
  [1]  1.729572e-05 -3.715927e-05 -7.860615e-06
  [4] -7.344127e-06 -1.739251e-05  5.139677e-06
  [7]  4.465275e-06 -6.186356e-05 -1.573240e-05
 [10]  1.642000e-07 -4.138122e-05  1.197398e-05
 [13]  2.971648e-05  1.928854e-05 -3.898240e-05
 [16]  2.529379e-05 -4.323282e-05  7.931323e-06
 [19] -8.576211e-07 -1.404878e-05 -5.775504e-05
 [22] -5.675964e-06 -1.615054e-05 -1.173680e-05
 [25] -5.732061e-05 -2.180526e-05 -4.471830e-05
 [28] -2.457135e-05  3.887581e-06 -4.691062e-05
 [31] -1.399816e-05 -2.128748e-05 -1.007771e-05
 [34] -1.648659e-05 -2.377591e-05 -3.757692e-06
 [37]  5.551087e-06 -1.177384e-05 -8.949698e-06
 [40]  2.804389e-06  7.922135e-06 -1.169350e-05
 [43]  2.674952e-05 -8.846469e-06 -3.416494e-06
 [46] -2.761855e-05 -6.186356e-05 -1.042239e-05
 [49] -5.226139e-05  3.121237e-05 -3.160226e-05
 [52] -1.739251e-05 -6.117518e-06 -2.962274e-05
 [55] -4.753742e-06  1.360054e-06 -3.006724e-05
 [58] -4.290463e-05 -4.034215e-05 -6.842842e-05
 [61] -4.999142e-05  3.493410e-05 -1.177384e-05
 [64]  1.426632e-05 -6.390825e-06 -1.668115e-05
 [67]  1.822556e-05 -5.896269e-05 -4.189638e-06
 [70] -7.708527e-06 -1.883925e-05 -9.929661e-06
 [73] -1.844299e-05  2.804389e-06 -4.034215e-05
 [76] -6.461821e-05 -4.545136e-05 -2.300112e-05
 [79]  2.570224e-05  3.878188e-06  2.510190e-05
 [82] -4.367071e-05 -1.702946e-06 -1.702260e-05
 [85] -1.050108e-05 -4.408547e-05 -2.942068e-05
 [88] -1.883925e-05  1.160394e-05 -9.811173e-06
 [91] -1.095042e-05  3.072011e-05 -1.548618e-05
 [94] -2.972262e-05  1.729572e-05  8.705455e-06
 [97] -1.810506e-05 -3.986050e-05 -2.364875e-05
[100] -3.052430e-05
```

Data lists of the same structure can take

7 How data lists are made

Up until now we have used an existing data list to illustrate the use of the `multitable` package. Although there are several ways to create data lists, there

is one way that provides the simplest framework for understanding the difference between variables and dimensions of replication, which is an important distinction to understand in order to use `multitable` most effectively.

Consider a data frame of species abundances counted at various sites.

```
> 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 data frame we have a data frame of environmental variables at each site and a data frame of traits for each species.

```
> environment
        sites temperature precipitation
1   subarctic           0            40
2 midlatitude          10            20
3 subtropical          20           100
4    tropical          50           150
5  equatorial          30           200
> trait
   species body.size metabolic.rate
1 capybara       140             20
2     moss         5              5
3  vampire       190              0
```

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 harsh and remote).

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; Fig. 5). 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, we could relate pH and temperature if they were both replicated along the same set of lakes. Similarly,

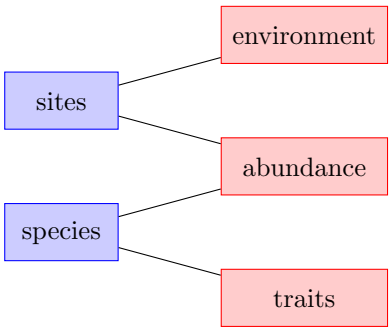

Figure 5: Bipartite graph of the multiple-table structure of data with a standard fourth-corner problem (Fig. 1). Dimensions of replication are in blue and tables are in red.

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 `dcast` function, which was inspired by the `acast` function in the `reshape2` package (Wickham, 2007).

```
> dl <- dlcast(list(abundance,environment,trait),
         dimids=c("sites","species"),
         fill=c(0,NA,NA)
 )
> dl
abundance:
---------
            capybara moss vampire
midlatitude        4    6       0
subtropical       10    0       1
tropical           8    9       0
equatorial         7    3       0
arctic             0    5       0
subarctic          0    0       0
Replicated along:  || sites || species ||


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

```
precipitation:
-------------
midlatitude subtropical    tropical  equatorial      arctic
         20         100         150         200          NA
  subarctic
         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 function takes three arguments: (1) a list of data frames, (2) a character vector, `dimids`, with the names identifying the dimensions of replication (i.e. the names of the columns shared between the tables), and (3) a vector, `fill`, with one element for each data frame giving the value with which to fill in any structural missing values. This last argument is particularly interesting, because we can fill missing abundances with zeros because those site-species combinations were not observed and `NA` values for the other tables. This data list can now be used in analyses.

It is quite possible that your data are not stored in data frames with columns for both dimensions of replication and variables; this is not a large concern, as the `multitable` package offers many ways to get your data into a data list (see vignette ????). However, we recommend that researchers at least consider what their data might look like in this format, because we believe that this format best illustrates concepts that will help make multiple-table data easier to understand, manage, and analyze. I now finish with some further elaboration of these concepts.

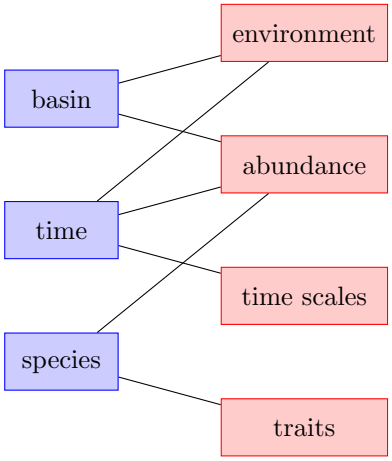

Figure 6: Bipartite graph of the Lac Croche data in Fig. 2.

8 Multiple-table concepts

Shared columns (i.e. dimensions of replication) between tables are expanded by `dlcast` into the dimensions of the arrays that are used to store each of the other columns (i.e. variables). For example, because the `abundance` table has two dimensions of replication, it is stored as a two-dimensional matrix in the resulting data frame; whereas, the `environment` and `trait` tables have only two dimensions of replication and so are stored in one-dimensional vectors.

The benefits of the distinction between dimensions of replication and variables, is that it provides a common framework for understanding both simple and more complex multiple-table data structures. In particular, the framework allows us to visualize the structure of complex data; for example the Lac Croche zooplankton community data (Fig. 2) has a structure given by Fig. 6. To store these data in a format amenable to `dlcast`, we would create one data frame for each of the groups of variables (red boxes) and add a column for each dimension of replication (blue boxes) associated with those variables.

With our data in this form, we can easily state the two requirements for using `data.list` objects: (1) every table must share at least one dimension of replication with at least one other table and (2) at least one table must be replicated along all of the dimensions present in the data set. The first criterion ensures that the tables will relate to each other; the second criterion ensures that some variables will be relatable to all other variables, a property that we feel is necessary for a response variable. We also see that we only need more than one table if some variables are not replicated along all of the dimensions.

9 Conclusion

The structure of `data.list` objects is sufficiently rich to give rise to a much wider variety of uses than I could cover here. However, this vignette was only intended to illustrate the basic features and concepts of the `multitable` package, and to justify its utility. My long-term goal with the `multitable` project in general is to make standard analyses in R (and beyond?) simpler to conduct on complex multiple-table data.

Acknowledgements

I thank Guillaume Guénard, Levi Waldron, Ben Bolker, and Philip Dixon for discussions and suggestions about software design.

References

Chambers, J. M. and T. J. Hastie, 1992. Statistical models in S. Wadsworth and Brooks, Pacific Grove, California.

Dolédec, S., D. Chessel, C. ter Braak, and S. Champely, 1996. Matching species traits to environmental variables: a new three-table ordination method. *Environmental and Ecological Statistics* **3**:143–166.

Dray, S. and P. Legendre, 2008. Testing the species traits-environment relationships: the fourth-corner problem revisited. *Ecology* **89**:3400–3412.

Ives, A. R. and M. R. Helmus, 2011. Generalized linear mixed models for phylogenetic analyses of community structure. *Ecological Monographs* **81**:511–523.

Legendre, P., R. Galzin, and M. L. Harmelin-Vivien, 1997. Relating behavior to habitat: solutions to the fourth-corner problem. *Ecology* **78**:547–562.

Leibold, M. A., E. P. Economo, and P. R. Peres-Neto, 2010. Metacommunity phylogenetics: separating the roles of environmental filters and historical biogeography. *Ecology Letters* **13**:1290–1299.

Pillar, V. D. and L. D. Duarte, 2010. A framework for metacommunity analysis of phylogenetic structure. *Ecology Letters* **13**:587–596.

Sólymos, P., 2009. Journal of statistical software. *Processing Ecological Data in R with the mefa Package* **29**:1–28.

Wickham, H., 2007. Reshaping data with the reshape package. *Journal of Statistical Software* **21**.