Trait-based ecology tools in R

Lars Götzenberger, Francesco de Bello, Andre TC Dias,
Marco Moretti, Matty P Berg, Carlos P Carmona

2021-01-17

Contents

1 – General Introduction

Welcome to the R material accompanying the reference textbook *"Handbook of Trait-Based Ecology: From Theory to R Tools"* (Cambridge University Press). The material we present here is a set of practicals using the statistical software R, designed to complement the book. Although it can be used without the book, the book does provide all the conceptual and mathematical ground on which this R material is built. This R material will hopefully provide you with all the technical knowledge you need to translate the theoretical foundations laid out in the book to actual data analyses. At the same time, the book will provide the ecological context for which the methods described in this manual are useful. Given the main target of the book, these techniques and analyses will mainly focus on handling and analyzing trait data, often in combination with community, environmental, and phylogenetic data, or any combination of these additional 'ingredients'.

1.1 Content and structure

We structured this document in a way so that R material chapters correspond to chapters in the book. So if we refer, for example, to R material Ch 6 you already know that this R material will contain approaches and methods that are connected to the topics contained in Chapter 6 of the book. We made this analogy in numbering as consistent as possible. As a consequence, since in the main book we do not provide specific methods in Chapter 1, 11 and 12, there is no R materials for these chapters. Some R material chapters might have subchapters that do not correspond directly with the subchapters in the book, so as to keep consecutive numbering for the subchapters. For instance, in the case of Chapter 5, in the R material we present two separate chapters for CWM 5.1 and FD 5.2, which are treated in subchapters 5.2 and 5.3 in the book. In R material Chapter 2, we introduce the package `traitor` that deals with missing trait data, a topic which (strictly speaking) is dealt with in Chapter 11 of the book. The following table gives you a quick overview of the contents of each R material.

R material	Relevant chapters from the reference book	Topics
2	2, 11	Data bases; missing data; synonymy
3	3, 6	Trait dissimilarity calculation
4	4	Species level response to the environment via multivariate analyses; boosted regression trees
5	5	Calculation of community weighted mean and functional diversity indices
6	6	Intra- vs. interspecific trait variability; trait probability distribution
7	7, 4	Null models; environmental filtering; competition
8	8, 2, 3, 5	Phylogenetic trees; phylogenetic signal; data imputation; phylogenetic diversity
9	9	Relationship between traits and ecosystem functions; net diversity effects
10	10, 9	Interaction niche; multitrophic interactions in the response-effect trait framework

Figure 1.1: R materials, their related chapters and covered topics

1.2 Prerequisites

We designed this manual for people with a basic knowledge of R and some basic statistical skills. Therefore, particularly for the material in the first chapters, we have kept things as simple as possible, with a very step-by-step approach in the R scripts and the statistical analyses considered. R was chosen as a common and open source platform in which many of the existing tools in trait-based ecology are being developed constantly. The methods described in our R material are often extensions of more general statistical univariate and multivariate methods, and we cannot introduce all of these here in depth. We therefore present here a list of resources that we find very useful in case you need to learn about or brush up your knowledge in general statistics and R language skills for the practicals in this R material.

> How to Install R on Windows, Mac OS X, and Ubuntu An overall introduction to R, from the very basics like data structures, data import, and installing packages to some basic statistics as linear regression and ANOVA.
>
> YaRrr! The Pirate's Guide to R Instructions how to install R and R studio for different Operating Systems. From the datacamp website.
>
> R for Ecologists Short introduction to R, especially data structures and types, and data import. By Dave Roberts.
>
> Analysis of community ecology data in R A comprehensive yet concise introduction to a wide array of multivariate methods and how to apply them in R. By David Zeleny.
>
> Phylogenetic Comparative Methods Tutorials Covers similar ground as in Chapter 8 of the R material but also provides some additional methods and approaches.

In general, it should be possible to run all code that we provide in these materials directly in R and its own user interface. However, we still recommend to install R Studio, as it is in many ways more convenient and practical to work with than with the interface provided by R. If you never heard about R Studio, or you are new to R entirely, you can check the set of features before you commit to installing it.

All the R material provided in this manual has been tested by the authors and, to a great extent, by a number of students which attended the courses we regularly organize. However, it might still happen that you will run into some errors and have to employ some troubleshooting, for which we apologize in advance. In most cases we expect that most of the problems will arise by using older R versions, and the corresponding problems with compatibility of the different packages we are using. As authors we are certainly willing to assist you to solve the main problems you might encounter when running the R materials in this manual. We are indeed very happy to solve all problems connected to possible mistakes, or unclear steps in our scripts and corresponding texts, so constructive suggestions are always extremely welcome. At the same time we kindly, but firmly, invite you to first contact some of your colleagues more expert in R, before contacting us, especially if the problems you are encountering are basically related to the ability to solve problems in R.

1.3 Scripts, packages, and additional functions and data

For each of the R materials described in this manual we provide R scripts which can be run, step-by-step, on your computer. In most cases, additional packages, data, and R-functions need to be loaded. In each of the R materials we first introduce the type of analyses that will be treated. Then we describe the R packages that will be needed, beside those already available by default. Finally we describe and import data which will be used. In some cases, some specific R functions that we, or others, have elaborated to run specific methods will be

needed. The data used and these ad-hoc functions are organized by chapters and available here.

As an important note, we cannot stress enough that the R tools described in this manual are only a selection of the many, and interesting, approaches existing in the literature from the field of trait-based ecology. For this reason, several of the tasks and questions we try to address with the proposed methods can be analyses with other useful tools as well. At the same time we think the selection of methods provides a robust 'entry pack' to the field of trait-based ecology and the diversity of methods and approaches existing in the field. As such, we think that the methods described will provide a solid basis for developing your analyses or improving even more the existing methods. In the future we certainly plan to include new developments and tools in the collection of provided materials. We thank you in advance for suggestions in this respect.

1.4 Some definitions and standards

In the R universe, there is an established and strict set of terms that is used for the different objects and data structures that can be encountered (mostly stemming from statistical and IT terminology). Sometimes, these terms are at odds and lead to some confusion with more loosely defined terms we use in ecology and in particular for ecological data. We therefore provide here a short glossary of terms that are somewhat ambiguous and could lead to confusion. At the same time, this section serves to establish some standards that we use across the R material, in particular regarding the orientation of data in rows and columns in our data objects. Beside this clarification we have tried hard in this manual to avoid ambiguous terms and explained as much as possible all objects used.

R terminology

- matrix: a two dimensional array of rows and columns. It can only contain one specific data type, e.g. numerical data. When trying to transform a data frame (see next point) that contains different data types into a matrix (with `as.matrix`), all data get coerced to be of the most basic data type. E.g. if the data frame contains columns with numerical numbers, and other columns with characters (i.e. text), the numerical columns will be turned into columns of the type character in the matrix.
- data frame: like a matrix, it is an array of rows and columns but, unlike a matrix, it can hold different types of data (e.g. numerical and categorical, with the latter being represented in R as characters or so-called factors) across the different columns.
- distance matrix: and object of class distance (`dist`). It usually includes information about the dissimilarity between pairs of objects (e.g. species). In the text it is often referred to as a distance matrix, although in R it is neither a matrix, nor a data frame. When displayed in the R console, it has triangular shape, because it only contains values in the cells below the diagonal. However, such a distance object can be transformed into a matrix with the `as.matrix` function (and back into a distance object with the `as.dist` function). Because there are the same number of columns and rows in the resulting matrix, this is a square matrix, with the same objects from 'left to right' as from 'top to bottom', i.e. the names of rows and columns will be identical. This square matrix contains the same data as the distance object, but twice. In most cases, the diagonal will contain only zero values, as the distances of objects with themselves are zero. At the same time, the diagonal is the axis along which data in the lower triangle is 'mirrored' to the upper triangle. See examples in Chapter 3 of the reference book.

Ecological terminology

- Community matrix (or 'site by species', or ´species x plot´ matrix): A table containing community data, i.e. the occurrence or abundance of species in different sampling units. This is also referred to as species composition or community composition data. For different packages there are different conventions regarding the R data structure the

community matrix should be in. This will be either a matrix, a data frame, or the possibility to provide either. The help of whatever function you are using should give you clear instructions about the data types and structures that it expects. Regarding the orientation of the community matrix, virtually all packages and functions taking a community matrix as input expect the species to be columns and the samples (sites, vegetation plots, pit fall traps, etc.) to be rows. However, especially in vegetation ecology, the community matrix can be represented with species as rows and sites/plots as columns (e.g. the vegetation data used in R materials Chapter 5). In such cases, the use of the transpose function (`t()`) will make the necessary '90 degree turn' of the data.

- Distance matrix (see also above): distance matrices are used very often in methods of ecological statistics, as very often we are interested in storing or representing some sort of similarity or dissimilarity (i.e. distance) between a set of objects. For instance, we might be interested how similar species are among each other based on their traits, which can be expressed as differences in trait values between species. Beta diversity measures are another good example; beta diversity expresses how different samples are with respect to their species composition, and we therefore obtain a beta diversity value for each pair of samples. The pairwise character of distance matrices becomes obvious in their structure, as the same objects (i.e. species or plots) are repeated in rows and columns. Therefore, sometimes also the term pairwise distance matrix is used, to highlight the fact that it contains, for example, trait distances between all possible pairs of species. The most important point in the context of working in R is that essential information contained in distance matrices can be stored and handled in two ways, as objects of class distance, or as a square matrix (see above under R terminology).
- Trait table (species x trait matrix): a table containing the traits we want to use for analyses. In R, a trait table will most often be in the form of a data frame, because this allows for having different types of variables (e.g. continuous and categorical) in the same object. Importantly, the species names will most often be expected to be the `row.names` of the data frame, and have to be following the same standard, spelling, etc. to be correctly matched to species in other data (community matrix, phylogeny).

1.5 Acknowledgements

Composing the reference text book, and this R material, has been made possible by the support of many institutions, including the Spanish National Research Council (CSIC, who hosts this page), the University of South Bohemia (USB), the Institute of Botany of the Czech Academy of Sciences (IBOT), the Vrije Universiteit Amsterdam, the Groningen Institute of Evolutionary Life Sciences, the University of Tartu (UT), the Swiss Federal Institute for Forest, Snow and Landscape Research (WSL) and the Federal University of Rio de Janeiro (UFRJ). Our families and colleagues have supported us in so many ways. Above all we thank Paulo Sousa for initiating all this in 2007 and keeping the trait course in Coimbra active. We sincerely thank all the students that have attended our courses over the years, for raising so many interesting questions, testing our R scripts and, with their doubtful remarks and lost expressions, showing where explanations have needed improving. We also thank Thomas Galland, Javier Puy, Maria Majeková, Anna Vojtko and Marta Gaia Sperandii, for reviewing earlier versions of the different sections in this manual. We are very grateful to Janine Märien, who made the illustrations that appear at the beginning of each chapter Finally, big thanks to Diego Trindade, who performed the final checks and compiled this document.

This document was made using R-Markdown from RStudio 1.3.1093 and R version 4.0.0 (2020-04-24)

2 – Trait Data

In this R material we will explore possibilities to obtain trait data from online databases and highlight what things to consider when using such data. Then we will explain how to handle the often occurring situation that databases are often not complete with respect to the list of species that we are searching trait data for (more information how to impute missing data can also be found in R material Ch 8.

While we mainly focus here on R packages, functions, and approaches that are connected to Chapter 2 of the reference textbook, we also treat one particular package (`traitor`) that can be seen as connecting concepts of Chapter 2 and Chapter 11. Chapter 11 mainly focuses on strategies of how to obtain trait data in the field in the best possible way. Package `traitor`, as we will see later, helps us identify species that we need to collect trait data for in the field, after we established for which and how many species trait information is missing.

2.1 Data

In the examples below data from the LEDA trait database is used, which contains various traits for the flora of North-West Europe. These data are freely available and relatively easy to access as .txt files from a website (https://uol.de/en/landeco/research/leda).

A second set of data will be used in combination with the package `traitor` introduced in section 2.5 of this R material. This data consists of community data from a wet meadow site in the Czech Republic called 'Ohrazeni' and trait data for the species occurring there.

2.2 Required packages and additional functions

As always, first make sure that the packages we are going to load are actually installed on your system, and install them if that is not the case yet.

The package `traitor` is not available through the standard CRAN website, but from a github repository. To install it, you first need to install an additional package (if you haven't already).

```
#install.packages("devtools")
library(devtools)
install_github("larsito/traitor")
```

Once installing traitor is taken care of, and all other packages are already installed, we can load them.

```
library(taxize)
library(Taxonstand)
```

```
## Loading required package: pbapply
```

```
library(traits)
```

```
## Registered S3 method overwritten by 'hoardr':
##   method           from
##   print.cache_info httr
```

```
library(traitor)
library(plyr)
```

2.3 Obtaining and handling data from the LEDA plant trait database

There is a multitude of ways in how we can obtain trait data regarding the physical location of the database and the type of data files, and how we get hold of them. For instance, data can be physically located on a server to which we don't have direct access, but first we need to make a request to an institution managing the data, often in the form of a species list (for example, the very comprehensive TRY database). For the example we use below, the data is more directly accessible, as downloadable .txt file from a dedicated internet website. Whatever the source, using data from databases or data sets of published papers does often require some necessary 'cleaning', as we will demonstrate below.

Here we will use a single trait from the LEDA database by downloading, through R, a single .txt file that contains data on canopy height, i.e. the height above the soil surface of the highest photosynthetically active leave(s). Using the function `download.file`, there are two paths that we need to specify. The first one, through the argument `url`, is the location of the text file on the internet (i.e. a link address). The `destfile` argument is used to specify the path to which to download the file, including the name of the file. In our case, we just specify the name of the .txt file and the destination will be automatically the current working directory (which is the default when no other path is specified before the file name). The function gives feedback and even a progress bar that tells how much of the file has been downloaded. In case of a decent internet connection this download should be rather quick and done in a few seconds.

```
urlLeda <- "https://uol.de/f/5/inst/biologie/ag/landeco/download/LEDA/Data_files/canopy_height.txt
download.file(url = urlLeda, destfile = "height_leda.txt")
```

Now that we have downloaded the text file, a very first thing to do is to open the file within R studio or a text editor of your choice and have a look at the content. You will see that there is a lot more than just the plain trait data!

At the very beginning, there is a big chunk of text we can ignore, as it is just returning some information about how and when this text file has been created. However, it is important to know what we need to 'skip' over from this first chunk when reading the file into R, as it would otherwise mess-up the import. With the `skip` argument set to 3, the number of lines that have to be skipped-over is defined (even if it might look more than 3 lines in your editor). The imported table should be saved as an object, here named `height_leda`. We also can see in the editor that semicolons are used to separate the columns and we have a row that contains column names, so we set the arguments `sep` and `header` accordingly. Additionally, we get rid of all so-called white space.

```
height_leda <- read.table("height_leda.txt", skip = 3, sep = ";", header = TRUE,
                          strip.white = TRUE)
```

White space in text files appears white on the screen, but is in reality not 'empty' for a computer, because they were created by using the space bar or tab key, i.e. any key on your keyboard that moves forward the cursor without actually typing any numbers, letters or symbols. E.g., to R (and most other programs that can process text information), "SLA" and " SLA " are not the same. Since such white space can slip in unintentionally (for example

when typing data in Excel sheets), it can be important to automatically delete them when importing data. White space is only stripped at the beginning and the end of an entry, so that space between words of the same entry stays intact. This is important as it prevents, for instance, that species names are collapsed into a single string of letters (i.e. "Acer campestre" will not become "Acercampestre" after importing data). The 'read.table' function by default changes all the really empty cells into NA values. Basically, the `read.table` function has roughly two dozens of arguments that can be used to fine-tune how the data get imported. We recommend to study the help file for that function, should you ever run into some specific trouble importing data.

To get an overview what the actual data table looks like, we opt to open it with the `View` function, which will open it in a sheet in an external window, which makes it better readable than displaying it in the R console.

```
View(height_leda)
```

There is maybe still more in the data table than you might have expected. In the most simple case, what a trait table contains are the names of the species and their trait values, either for a single or possibly a multitude of different traits. Although this is the minimal information we need, this supplementary data can be useful, even if it makes live a bit harder when preparing the data. There is, among other things, information on a mean trait value, minimum and maximum values, the sample size these measures are based on, references in case data was extracted from published literature or already available data bases, and the methodology behind the measurement. Potentially, all this additional information can be used to optimize our data, for instance when data obtained from "laboratory/greenhouse/garden experiment" (see the 'general.method' column) should not be included because these plants might not have been grown under "natural" conditions.

In the data table there appear to be two different types of mean values for the species. The first one is in column 3 and is named 'single.value..m.', while the second one named 'mean.CH..m.' can be found before the maximum and minimum values. Why these two columns that appear to have the same information of interest, and moreover, why is one of them mostly filled with NAs? The answer to these questions is that within LEDA, most of the data on canopy height stem from two data sources, which are themselves databases. The first one, with the majority of entries, is BIOPOP, and the second one is "ECOFLORA - database of the ecological flora of the British Isles". For both, most of the data come in the form of maximum and minimum values. The creators of the LEDA database decided to take the "mean" of these maximum and minimum values to present an average value of canopy height per species, but since this value is not an actual statistical mean (i.e. the average of N measurements of height of N individuals), an additional column was created ('single.value..m.'). This column therefore is a mixture of "fake" and also "real" mean values, the latter coming from height data that is actually based on measured individuals.

All these intricacies, which in the case of LEDA are even trait-dependent (i.e. for other traits other "rules" apply), make it clear how important it is that we have some sort of key that helps us understand the data. As a minimum, we should have some metadata (i.e. data about the data) that informs us about used abbreviations, applied methodologies, etc.. Often, such metadata are stored as additional .txt or Excel files that accompany the actual trait data files. In the case of LEDA, we have the luxury that it has its own trait standards, i.e. a very detailed description of how the different traits were measured, and how they are represented in the database. This info can be downloaded from the same webpage that is used to download the data (https://uol.de/en/landeco/research/leda/standards/).

A last thing that we have to recognize before we extract the data is the fact that for many species there is more than a single record for height in LEDA. In most of the cases, this is because information for the same species was found across different sources, and all of them were entered into the database. If we are interested in obtaining a single value per species, we need to aggregate the data at the species level. A straight forward function in R to use is `aggregate`. The function `aggregate` takes as first argument the variable or object you would like to aggregate (`single.value..m.`), and the second argument specifies the variable

that you want to use as aggregation level. This will usually be a categorical variable, as also in our case, i.e. `SBS.name` (species identity). For the function `aggregate` this second argument should be provided as a list, which is easily achieved by using the `list` function. The last necessary argument (`FUN`) determines which operation we want the aggregation to perform. As we want the average value of height for a species, we use `mean`.

```
height_leda_agg <- aggregate(height_leda$single.value..m.,
                             list(height_leda$SBS.name), FUN = mean)
head(height_leda_agg)
```

```
##                Group.1        x
## 1           Abies alba 50.00000
## 2        Abies grandis 50.00000
## 3   Abies nordmanniana 30.00000
## 4 Acaena anserinifolia  0.15000
## 5       Acer campestre 13.42143
## 6  Acer monspessulanum 11.00000
```

This creates the requested species by single trait table. Technically not difficult to achieve, but you see that it is necessary to have a very good idea about the data before anything is done with it. Whatever the type of organisms you are working with, unless you already get a "clean" species by trait table, you will have to make decisions how and what to select and aggregate, based on biological knowledge of the species and trait at hand. No database, no matter how fancy and large, can make this process avoidable, it can only provide you the necessary information.

2.4 Handling different taxonomic concepts across data sets - dealing with synonymies

When using species-related data there is no guarantee that a given species has the same name in our data and in the trait data we plan to use. Taxonomists all over the world are constantly discovering new species and working on entangling, delimiting, and describing the often complicated taxonomic relationships between them. Given the tremendous diversity on our planet, it is a task of astronomic proportions to sort and name all organisms, and then also to constantly update and disseminate this information to all the researchers who are not taxonomists themselves, but depend on being able to define and name accurately what species they are working with. In many cases, the working taxonomy that we use to name "our" species is based on regional or national faunas or floras, or some other source that is considered to be a "standard list" of species and their names. At least within these geographically defined borders we can communicate with our colleagues, and know that we both mean the same thing when we say *Abies alba* (silver fir) or *Parus major* (great tit). However, more often than not, taxonomies are not used consistently, and different "taxonomic concepts" are used in different parts of the world, or across different generations of researchers. This then often results in a situation where a species named XY in the list of species you are working with, is not called the same in another set of data, e.g. in the trait data available from a trait data base. The most frequently occurring practical problem in such situations is that we fail to *match* correctly species in our list to the species in external trait data via their names. As a consequence, trait data cannot be obtained, even though the species is part of the database. For some taxonomic groups, there is the possibility to access taxonomic databases that serve as a standard reference. Generally, such databases provide the possibility to look up a particular species name and retrieve relevant information about its status and synonyms. This service is offered through websites. Various R packages provide the possibility to query these websites directly from R, instead of having to switch to your internet browser of choice.

The package `taxize` gives access to a large number of taxon databases. It offers a great flexibility in the kind of information you can retrieve from taxon databases. The number of functions is a bit overwhelming at first, but they are basically a combination of two parts. The first part is an abbreviation of the taxon database one wants to query, and the second part the set of specific functions used to obtain information from each database. The kind of information one can extract differs between databases. Below we discuss some of the functionalities provided in `taxize`.

In many cases, we might as a first check want to see if we are actually using species names that are at all in accordance with a particular taxonomic data source. This can be done with the `resolve` function.

```
resolve("Vulpes vulpes")
```

```
## $gnr
## # A tibble: 41 x 5
##    user_supplied_na~ submitted_name matched_name data_source_title         score
##  * <chr>             <chr>          <chr>        <chr>                    <dbl>
##  1 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ NCBI                     0.988
##  2 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ Freebase                 0.988
##  3 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ Encyclopedia of Life     0.988
##  4 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ China: Yunnan, Southern ~ 0.988
##  5 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ AnAge                    0.988
##  6 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ Global Invasive Species ~ 0.988
##  7 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ ARKive                   0.988
##  8 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ CU*STAR                  0.988
##  9 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ nlbif                    0.988
## 10 Vulpes vulpes     Vulpes vulpes  Vulpes vulp~ Index to Organism Names  0.988
## # ... with 31 more rows
```

The data frame generated by the function is relatively large with a lot of entries. In fact, what the `resolve` function does by default, is to use the "Global Name Resolver" which itself then queries a lot of different taxon data sources, which are not necessarily searchable from within `taxize`. So, while we can be pretty confident that we searched for a species name that is actually used as such over a wide range of sources, we do not get much more information than that, apart from knowing in which databases a name is used. Some taxonomic data source that are available through `taxize` contain information about specific taxonomic groups. For example, when working with marine organisms we can use `taxize` to query the "World Register of Marine Species" (WoRMS).

As a first step, you can use `get_wormsid` to obtain the ID used in WoRMS to identify the species you are looking for. Such IDs are used in all databases to uniquely identify entries within them. The function returns a single number with some additional attributes to it.

```
get_wormsid("Monodon monoceros")
```

```
## ==  1 queries  ===============

##
## Retrieving data for taxon 'Monodon monoceros'

##   Found:  Monodon monoceros
## ==  Results  =================
##
##  Total: 1
##  Found: 1
##  Not Found: 0
```

```
## [1] "137116"
## attr(,"class")
## [1] "wormsid"
## attr(,"match")
## [1] "found"
## attr(,"multiple_matches")
## [1] FALSE
## attr(,"pattern_match")
## [1] FALSE
## attr(,"uri")
## [1] "http://www.marinespecies.org/aphia.php?p=taxdetails&id=137116"
```

This ID number is used to call additional functions that work on the WoRMS database. For instance, to get an overview of the taxonomic classification of the species starting from the kingdom.

```
classification(get_wormsid("Monodon monoceros"))
```

```
## ==  1 queries  ===============

##
## Retrieving data for taxon 'Monodon monoceros'

##   Found:  Monodon monoceros
## ==  Results  =================
##
##  Total: 1
##  Found: 1
##  Not Found: 0

## $`137116`
## # A tibble: 14 x 3
##    name              rank            id
##    <chr>             <chr>        <int>
##  1 Animalia          Kingdom          2
##  2 Chordata          Phylum        1821
##  3 Vertebrata        Subphylum   146419
##  4 Gnathostomata     Superclass    1828
##  5 Tetrapoda         Superclass    1831
##  6 Mammalia          Class         1837
##  7 Theria            Subclass    380416
##  8 Cetartiodactyla   Order       370511
##  9 Cetancodonta      Suborder    370545
## 10 Cetacea           Infraorder    2688
## 11 Odontoceti        Superfamily 148723
## 12 Monodontidae      Family      136983
## 13 Monodon           Genus       137030
## 14 Monodon monoceros Species     137116
##
## attr(,"class")
## [1] "classification"
## attr(,"db")
## [1] "worms"
```

Or to get a data frame that shows all the different synonyms for that species, including the authorship of these synonyms, info on the classification, etc.

```
synonyms(get_wormsid("Monodon monoceros"))
```

```
## ==  1 queries  ===============

##
## Retrieving data for taxon 'Monodon monoceros'

##   Found:  Monodon monoceros
## ==  Results  =================
##
##  Total: 1
##  Found: 1
##  Not Found: 0

## $`137116`
## # A tibble: 9 x 27
##   AphiaID url   scientificname authority status unacceptreason taxonRankID rank
##     <int> <chr> <chr>          <chr>     <chr>  <chr>                <int> <chr>
## 1  384243 http~ Ceratodon mon~ Pallas, ~ unacc~ synonym                220 Spec~
## 2  384244 http~ Ceratodontis ~ Brisson,~ unacc~ synonym                220 Spec~
## 3  384257 http~ Monodon micro~ Fleming,~ unacc~ synonym                220 Spec~
## 4  384258 http~ Monodon narhv~ Borowski~ unacc~ synonym                220 Spec~
## 5  384259 http~ Monodon narhw~ Blumenba~ unacc~ synonym                220 Spec~
## 6  384260 http~ Narwalus ande~ Lacépède~ unacc~ synonym                220 Spec~
## 7  384261 http~ Narwalus micr~ Lacépède~ unacc~ synonym                220 Spec~
## 8  384262 http~ Narwalus vulg~ Lacépède~ unacc~ synonym                220 Spec~
## 9  384263 http~ Tachynices me~ Brookes,~ unacc~ synonym                220 Spec~
## # ... with 19 more variables: valid_AphiaID <int>, valid_name <chr>,
## #   valid_authority <chr>, parentNameUsageID <int>, kingdom <chr>,
## #   phylum <chr>, class <chr>, order <chr>, family <chr>, genus <chr>,
## #   citation <chr>, lsid <chr>, isMarine <int>, isBrackish <int>,
## #   isFreshwater <int>, isTerrestrial <int>, isExtinct <lgl>, match_type <chr>,
## #   modified <chr>
```

As mentioned above, the information that the function `taxize` will give depends on which database is queried and it doesn't offer the most straightforward solutions for resolving synonym problems. One helpful package in that respect is `Taxonstand`, though it comes with the limitation that it only takes care of plant species. `Taxonstand` also provides an interface to a taxon database, namely The Plant List. There are only two functions in `Taxonstand`, which have the same result, just that function `TPL` can search an entire species list, whereas `TPLck` only searches for a single species. A strong feature of `Taxonstand` is that it provides detailed output of species on their taxonomic status in a tabular form:

```
Taxonstand::TPL(c(sp = "Elymus repens"))
```

```
##            Taxon  Genus Hybrid.marker Species Abbrev Infraspecific.rank
## sp Elymus repens Elymus                repens   <NA>               <NA>
##    Infraspecific Authority         ID Plant.Name.Index TPL.version
## sp                          kew-411505              TRUE         1.1
##    Taxonomic.status  Family New.Genus New.Hybrid.marker New.Species
## sp         Accepted Poaceae    Elymus                        repens
##    New.Infraspecific.rank New.Infraspecific New.Authority     New.ID
## sp                                          (L.) Gould kew-411505
##    New.Taxonomic.status  Typo WFormat Higher.level       Date Tax_res
## sp             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
```

In column 'Taxon' we see what name we actually submitted to be queried on The Plant List (i.e. as defined in the `sp` argument), followed by a number of columns that break-up the name, of which only 'Genus' and 'Species' are filled, because our species name didn't contain any other parts, such as the authors name, or a particular subspecies. Importantly, the column 'Plant.Name.Index' indicates that the species was indeed found in the database, and from 'Taxonomic.status' that it is an accepted name (i.e. not a synonym). The species belongs to the 'Family' Poaceae and has the database internal 'ID' kew-411505. In the following columns that the taxonomic splitting of the name is repeated, but now each column name starts with 'New.'. Since our species name is the accepted name, the species name is once more repeated here. Also, there was no typo in our species name. TPL can handle small typos in the epithet to some degree, but not in the genus name. Let's demonstrate the last point.

```
Taxonstand::TPL(c(sp = "Elymus repes"))
```

```
##           Taxon  Genus Hybrid.marker Species Abbrev Infraspecific.rank
## sp Elymus repes Elymus                repes   <NA>               <NA>
##    Infraspecific Authority         ID Plant.Name.Index TPL.version
## sp                          kew-411505             TRUE         1.1
##    Taxonomic.status  Family New.Genus New.Hybrid.marker New.Species
## sp         Accepted Poaceae    Elymus                       repens
##    New.Infraspecific.rank New.Infraspecific New.Authority     New.ID
## sp                                          (L.) Gould kew-411505
##    New.Taxonomic.status Typo WFormat Higher.level       Date Tax_res
## sp             Accepted TRUE   FALSE        FALSE 2021-01-17 Species
```

By what is generally called "fuzzy matching" of character variables, `TPL` is still able to detect our species correctly, although one letter was missed. However, when the differences in the names are too large, it will fail to detect the species in the database.

```
Taxonstand::TPL(c(sp = "Elymus reptans"))
```

```
##             Taxon  Genus Hybrid.marker Species Abbrev Infraspecific.rank
## sp Elymus reptans Elymus               reptans   <NA>               <NA>
##    Infraspecific Authority ID Plant.Name.Index TPL.version Taxonomic.status
## sp                                      FALSE         1.1
##    Family New.Genus New.Hybrid.marker New.Species New.Infraspecific.rank
## sp          Elymus                       reptans                     NA
##    New.Infraspecific New.Authority New.ID New.Taxonomic.status  Typo WFormat
## sp                                                        FALSE   FALSE
##    Higher.level       Date Tax_res
## sp        FALSE 2021-01-17   Genus
```

When a species name in The Plant List is a synonym, TPL will tell us the accepted name of that species. We demonstrate that here with the "old" species name of *Elymus reptans*.

```
Taxonstand::TPL(c(sp = "Agropyron repens"))
```

```
##               Taxon     Genus Hybrid.marker Species Abbrev Infraspecific.rank
## sp Agropyron repens Agropyron                repens   <NA>               <NA>
##    Infraspecific Authority         ID Plant.Name.Index TPL.version
## sp                          kew-388761             TRUE         1.1
##    Taxonomic.status  Family New.Genus New.Hybrid.marker New.Species
## sp          Synonym Poaceae    Elymus                       repens
##    New.Infraspecific.rank New.Infraspecific New.Authority     New.ID
## sp                                          (L.) Gould kew-411505
##    New.Taxonomic.status  Typo WFormat Higher.level       Date Tax_res
## sp             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
```

Compared to the previous search results, the `Taxonomic.status` is now `Synonym`, but `TPL` was able to connect the synonym to the accepted species name. The use of `Taxonstand` and its `TPL` function to translate our species list into accepted species names is very helpful. It enables us to match different species-based data tables, even when the different tables contain the same species but with a different name or with (a reasonable degree of) misspellings. We demonstrate this with an artificial data set. First, we create a trait data table with four plant species, with names that are not all correct. The correct names of the species used in the example are *Abies alba* (typo), *Bellis dubia* (synonym), *Helianthus annua* (misspelling), while *Cocos nucifera* is neither misspelled nor a synonym.

```
height <- data.frame(species = c("Abies alpa", "Chrysanthemum bellidiflorum",
                                 "Helianthus annua", "Cocos nucifera"),
                     height = c(45, 0.1, 1.9, 28))
```

Then we construct presence/absence community data, that *should* include the same species, but compared to our trait table from above, has all the correct names.

```
com <- matrix(c(1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0), 4, 4)
colnames(com) <- c("Abies alba", "Bellis dubia", "Helianthus annuus",
                   "Cocos nucifera")
rownames(com) <- c("community 1", "community 2", "community 3", "community 4")
com
```

```
##             Abies alba Bellis dubia Helianthus annuus Cocos nucifera
## community 1          1            0                 0              1
## community 2          1            1                 0              0
## community 3          0            1                 1              1
## community 4          0            1                 1              0
```

The first step to match the community data against the trait data is to see if the species names in both data sets are the same. Two useful functions in R, `intersect` and `setdiff`, can be used. `intersect` gives us the intersection (or union) of two variables, i.e. the elements in two variables that are present in both those variables (i.e., the intersection of variables {a, b, c} and {c, d, e} would be {c}). Since we want to obtain the set of common species names across the two data `sets` (community and traits) we can use `intersect` on the column names of the community and the `species` column of the trait data.

```
intersect(colnames(com), height$species)
```

```
## [1] "Cocos nucifera"
```

Cocos nucifera is the only species present in both data sets. However, all four species are present in both data sets, but due to synonymy and misspelling they are not matched. With only four species this is easy to see, but if your data contains dozen or even hundreds of species it becomes difficult. This is where the `setdiff` function comes in handy, as it helps us to detect which species are present in one data set, but not the other data set.

```
setdiff(colnames(com), height$species)
```

```
## [1] "Abies alba"        "Bellis dubia"      "Helianthus annuus"
```

Notice that while `intersect` gives the same result no matter which species name variable you put as the first argument, this is not true for the `setdiff` function. It always returns the species that are present in the first, but not in the second variable. If you switch them around the result is different.

```
setdiff(height$species, colnames(com))
```

```
## [1] "Abies alpa"                  "Chrysanthemum bellidiflorum"
## [3] "Helianthus annua"
```

Now the species names in both data sets are checked and we have found a mismatch in species names, but how to resolve this problem? `TPL` to the rescue! For both species names variables we send a query to the plant list through the `TPL` function, which will return for both the kind of table we explored above.

```
tpl_com <- TPL(colnames(com))
tpl_com
```

```
##               Taxon      Genus Hybrid.marker  Species Abbrev Infraspecific.rank
## 1        Abies alba      Abies                   alba   <NA>               <NA>
## 2      Bellis dubia     Bellis                  dubia   <NA>               <NA>
## 3 Helianthus annuus Helianthus                 annuus   <NA>               <NA>
## 4    Cocos nucifera      Cocos                nucifera   <NA>               <NA>
##   Infraspecific Authority          ID Plant.Name.Index TPL.version
## 1                         kew-2609691             TRUE         1.1
## 2                           gcc-92513             TRUE         1.1
## 3                           gcc-44475             TRUE         1.1
## 4                           kew-44645             TRUE         1.1
##   Taxonomic.status     Family  New.Genus New.Hybrid.marker New.Species
## 1         Accepted   Pinaceae      Abies                         alba
## 2         Accepted Compositae     Bellis                        dubia
## 3         Accepted Compositae Helianthus                       annuus
## 4         Accepted  Arecaceae      Cocos                      nucifera
##   New.Infraspecific.rank New.Infraspecific New.Authority      New.ID
## 1                                             Mill. kew-2609691
## 2                                           Spreng.   gcc-92513
## 3                                                L.   gcc-44475
## 4                                                L.   kew-44645
##   New.Taxonomic.status  Typo WFormat Higher.level       Date Tax_res
## 1             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
## 2             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
## 3             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
## 4             Accepted FALSE   FALSE        FALSE 2021-01-17 Species
```

```
tpl_height <- TPL(height$species)
tpl_height
```

```
##                       Taxon         Genus Hybrid.marker       Species Abbrev
## 1                 Abies alpa         Abies                        alpa   <NA>
## 2 Chrysanthemum bellidiflorum Chrysanthemum               bellidiflorum   <NA>
## 3            Helianthus annua    Helianthus                      annua   <NA>
## 4              Cocos nucifera         Cocos                    nucifera   <NA>
##   Infraspecific.rank Infraspecific Authority          ID Plant.Name.Index
## 1               <NA>                         kew-2609691             TRUE
## 2               <NA>                           gcc-77312             TRUE
## 3               <NA>                           gcc-44475             TRUE
## 4               <NA>                           kew-44645             TRUE
##   TPL.version Taxonomic.status     Family  New.Genus New.Hybrid.marker
## 1         1.1         Accepted   Pinaceae      Abies
## 2         1.1          Synonym Compositae     Bellis
## 3         1.1         Accepted Compositae Helianthus
```

```
## 4         1.1         Accepted  Arecaceae      Cocos
##   New.Species New.Infraspecific.rank New.Infraspecific New.Authority
## 1        alba                                                 Mill.
## 2       dubia                                               Spreng.
## 3      annuus                                                    L.
## 4    nucifera                                                    L.
##        New.ID New.Taxonomic.status  Typo WFormat Higher.level       Date
## 1 kew-2609691             Accepted  TRUE   FALSE        FALSE 2021-01-17
## 2   gcc-92513             Accepted FALSE   FALSE        FALSE 2021-01-17
## 3   gcc-44475             Accepted  TRUE   FALSE        FALSE 2021-01-17
## 4   kew-44645             Accepted FALSE   FALSE        FALSE 2021-01-17
##   Tax_res
## 1 Species
## 2 Species
## 3 Species
## 4 Species
```

Inspecting the outputs in the tables returned by the TPL function shows two different kinds of "mistakes" in the species names. First, *Abies alba*, was misspelled as *Abies alpa* and *Helianthus annuus* as *Helianthus annua* (sunflower) in the height data set. The `Typo` column indicated which species gave the problem. Second, *Chrysanthemum bellidiflorum* in the trait data set is a synonym of *Bellis dubia*, the name that occurs in the community data set. This information is retrieved from the `Taxonomic.status` column in `tpl_height`. To be able to link the two data sets, the problematic species names have to be translated into the proper names. To do this, the variable New.ID, which contains a unique but cryptic identifier for each species, or a combination of the `New.Genus` and `New.Species` variables can be used. We prefer to take the `New.Genus` and `New.Species` option as it is less cryptic. Note that to obtain the species names we need to combine (i.e. using `paste`) genus and species names. If there are species with hybrid markers (x) or sub-species level names in the data set this should be included in the `paste` command as well.

```
new_species_names_height <- paste(tpl_height$New.Genus, tpl_height$New.Species)
new_species_names_com <- paste(tpl_com$New.Genus, tpl_com$New.Species)
```

In this particular case one new name variable could have been created, because in both data sets the species occur in the same order. However, often this will not be the case (or at least you are not sure if it is the case), so it is a good habit of always doing this for both data sets separately. Now the names can be added or substituted. Using once more the `intersect` function, all species names now match.

```
height$new_species_names_height <- new_species_names_height
colnames(com) <- new_species_names_com
intersect(colnames(com), height$new_species_names_height)
```

```
## [1] "Abies alba"        "Bellis dubia"      "Helianthus annuus"
## [4] "Cocos nucifera"
```

In this example trait data and community data are matched, based on species occurring in both data sets. Of course, the same problems might occur if different trait data sets from different sources are combined, or the names of different community data sets are compared. But the principle stays the same: first check the species names for all data sets, substitute incorrect names, and then match/combine the data sets based on normalized species names.

As mentioned earlier, `Taxonstand` and its `TPL` function are a great way to deal with synonymy and typos in species names. We are not aware of packages that provide equally capable interfaces to other taxon databases than the plant list. Nevertheless, this section

hopefully demonstrated the importance of taking proper care of issues surrounding species names if you want to optimize the amount of information retrieved from data bases, or more generally the number of species matches across different sets of data.

2.5 Using the traitor package to support your trait sampling campaign

Missing data can affect the outcome of trait analyses, especially when the trait data is missing for species that are abundant in a community. How many NAs we can 'afford' in our data and still consider the results of the analysis to be correct is context dependent, but one suggested rule of thumb is that not more than 20% of the total abundance or biomass (see among others Pakeman and Quested (2007) but also Chapter 11.5 in the reference book). One strategy to mitigate the effect of missing data is to quantify how much information is missed, followed by measuring the traits of particular species. The package 'traitor' can be used to prioritize which species should be measured (Májeková et al., 2016) and we start with loading the library along with a data set that is provided with the packages.

```
library(traitor)
data(ohrazeni)
```

The function that we want to use for our purpose is called `sampleSpecies`, and its most important output is to tell us what species we should collect traits for given that we do not have complete trait data. It has four arguments; the argument `com` provides the community data, while `avail_sp` gives information about species from the community for which trait data is available, and for which not. Based on these two data `sampleSpecies` suggests which species in our community we should sample. The argument `thresh` defines the minimum threshold of summed abundance of species for which we want trait data. For instance, setting this threshold at 80% of total abundance, and assuming that in a given plot there is trait data for the species that make up 70% of the relative abundance, we need to sample the traits of as many species as necessary for an additional 10% of the total abundance. This could be a single species or more than one species, depending on their abundance. To understand the fourth argument, `sequential`, we have to introduce "pool-wise" (Scenario 1) and "plot-wise" (Scenario 2) scenarios, that are automatically applied by `sampleSpecies`.

Imagine a community matrix containing three plots. With the information from avail_sp, we can calculate for each single plot which species need to be sampled to reach the 80% total abundance threshold, as defined by argument `thresh`. Because the species composition will differ between the samples, `sampleSpecies` might suggest different species in order to reach the 80% in each of the samples. Let's look at this with some simple toy data. We set the sequential argument to FALSE for now, and will come back to its meaning later.

```
# 1. Create a toy community with 3 samples (rows) and 5 species (columns)
com_toy <- matrix(c(50, 40, 30, 20, 0, 30, 0, 10, 0, 5, 10, 0, 0, 0, 5), 3, 5)
# 2. Give "names" sp1 to sp5 to the species
colnames(com_toy) <- as.character(paste("sp", c(1:5)))
# 3. create a vector that defines for which species there is trait data (1) and
# for which there isn't (0)
avail <- c(1, 0, 1, 0, 1)
# 4. run the sampleSpecies function on the toy data
sampleSpecies(com = com_toy, avail_sp = avail, thresh = 0.8, sequential = FALSE)
```

```
## Warning: avail_sp is without names. Species are assumed to be in the same order
## as in community data.
```

```
## $Original
## $Original$com_available
##      sp 1 sp 3 sp 5
## [1,]   50    0    0
## [2,]   40   10    0
## [3,]   30    0    5
##
## $Original$available_pool
## [1] 0.675
##
## $Original$available_plot
## [1] 0.6666667 0.8333333 0.5384615
##
##
## $Scenario1
## $Scenario1$sample_species
## [1] "sp 2"
##
## $Scenario1$com_available
##      sp 1 sp 2 sp 3 sp 4 sp 5
## [1,]   50   20    0   NA    0
## [2,]   40    0   10   NA    0
## [3,]   30   30    0   NA    5
##
## $Scenario1$available_pool
## [1] 0.925
##
## $Scenario1$available_plot
## [1] 0.9333333 0.8333333 1.0000000
##
##
## $Scenario2
## $Scenario2$sample_species
## $Scenario2$sample_species[[1]]
## [1] "sp 2"
##
## $Scenario2$sample_species[[2]]
## [1] "none"
##
## $Scenario2$sample_species[[3]]
## [1] "sp 2"
##
##
## $Scenario2$com_available
##      sp 1 sp 2 sp 3 sp 4 sp 5
## [1,]   50   20    0   NA    0
## [2,]   40   NA   10   NA    0
## [3,]   30   30    0   NA    5
##
## $Scenario2$available_pool
## [1] 0.925
##
## $Scenario2$available_plot
## [1] 0.9333333 0.8333333 1.0000000
```

The output from the function is stored in a list with three elements, named `Original`, `Scenario1`, and `Scenario2`, respectively. The element `Original` contains a community data matrix, showing only the species for which there is trait data, so that in our case it only has

three species (`sp1`, `sp3`, and `sp5`) instead of all five species. Species 1, 3 and 5 together make up 67.5 % of the total abundance in the overall study (see `Original$available_pool`). Only the second plot would meet the 80% total abundance threshold without sampling additional species (see `Original$available_plot`).

The elements Scenario1 and Scenario2, each with an additional entry, specifies which species we should sample under the given scenario. To reach the 80% abundance threshold under the pool-wise scenario, species 2 should be sampled. If so, with this species in the data 92.5% of the total abundance is covered (see `Scenario1$available_pool`). That is more than needed because the function selects the most abundant species that helps to reach the threshold value, maximizing the benefit we obtain from sampling the least amount of additional species.

In the community data matrix under `$com_available` we now see the community represented as if we already sampled that species number 2, i.e. we can use it's abundance data in the plots to calculate functional diversity etc. Note also that, even though we have not yet considered individual plots to figure out which species to sample traits for (i.e. Scenario 2), we can see already that also the species abundances at the plot level (`$Scenario1$available_plot`) have all surpassed the 80%, so we hit two birds with one stone with using only Scenario 1. This is also the reason why the output in the third element `$Scenario2` is identical to the one from Scenario 1, with some small exceptions. For the plot-wise scenario, we get suggestions for each single plot which species to sample. The function suggest to measure species 2 for samples 1 and 3, but "none" for sample 2. In sample 2, 83% of total abundance is already represented by species with trait data.

We proceed by making a small change to the `avail` object we created, to see a result where Scenario 1 and Scenario 2 actually differ, by defining one more species that does not have trait data.

```
# Create a vector that defines trait data is missing for one additional species
avail2 <- c(1, 0, 0, 0, 1)
# Rerun the sampleSpecies function on the toy data
sampleSpecies(com = com_toy, avail_sp = avail2, thresh = 0.8, sequential = FALSE)
```

```
## Warning: avail_sp is without names. Species are assumed to be in the same order
## as in community data.
```

```
## $Original
## $Original$com_available
##      sp 1 sp 5
## [1,]   50    0
## [2,]   40    0
## [3,]   30    5
##
## $Original$available_pool
## [1] 0.625
##
## $Original$available_plot
## [1] 0.6666667 0.6666667 0.5384615
##
##
## $Scenario1
## $Scenario1$sample_species
## [1] "sp 2"
##
## $Scenario1$com_available
##      sp 1 sp 2 sp 3 sp 4 sp 5
## [1,]   50   20   NA   NA    0
## [2,]   40    0   NA   NA    0
```

```
## [3,]   30   30   NA   NA    5
##
## $Scenario1$available_pool
## [1] 0.875
##
## $Scenario1$available_plot
## [1] 0.9333333 0.6666667 1.0000000
##
##
## $Scenario2
## $Scenario2$sample_species
## $Scenario2$sample_species[[1]]
## [1] "sp 2"
##
## $Scenario2$sample_species[[2]]
## [1] "sp 4"
##
## $Scenario2$sample_species[[3]]
## [1] "sp 2"
##
##
## $Scenario2$com_available
##      sp 1 sp 2 sp 3 sp 4 sp 5
## [1,]   50   20   NA   NA    0
## [2,]   40   NA   NA   10    0
## [3,]   30   30   NA   NA    5
##
## $Scenario2$available_pool
## [1] 0.925
##
## $Scenario2$available_plot
## [1] 0.9333333 0.8333333 1.0000000
```

We are now down to 65% of relative abundance that is covered by species with trait data at the pool level, and almost at only a half (54%) in the third plot. It certainly wouldn't be the most reliable to calculate any trait-based diversity or composition index based on data that represents such a small proportion of the sampled vegetation. With the pool-wise scenario , we can remedy this situation, as earlier, by sampling traits for species 2, which meets our threshold of 80%, but with a lower total relative abundance than in the first example (87.5% vs 92.5%). Under this scenario, the second plot would still have only data for species that make up two thirds of plot relative abundance. This is where Scenario2 comes into effect. For the second plot, it suggests to sample species 4 instead of species 2, which is the species to be sampled for plots 1 and 3. This is because species 2 isn't at all present in sample 2, so it is not helping us to reach the threshold for that plot. Species 4 on the other hand makes up 17% of the total abundance of 60 in that plot, and adding that to the 67% or relative abundance of species 1 we reach the 80% threshold.

Finally , let's turn to the `sequential` argument. When `sequential` is set to `TRUE` the function first determines which species need to be sampled to reach the threshold according to Scenario 1, i.e. which species need to be sampled to ensure that the overall (pooled) abundance covered is at least the value set by `thresh`. However, since this leads to the individual plots not necessarily reaching this threshold (as we have seen above), subsequently species are determined by Scenario 2 (i.e. plot-wise) that need to be sampled in addition to the ones determined by Scenario 1, in order to reach the threshold for each of the plots. Such a sampling strategy would be appropriate if one needs community trait values, i.e. it does not matter which individual plot the trait measurements come from. In fact, they could even be from individual measured outside of the sample. It can also occur that the threshold is already reached for each plot with Scenario 1 (as in our first example), in which case a

warning is shown and the value for Scenario 2 in the output is `NULL`, but only if argument sequential is indeed set to `TRUE`.

```
sampleSpecies(com = com_toy, avail_sp = avail, thresh = 0.8, sequential = TRUE)
```

```
## Warning: avail_sp is without names. Species are assumed to be in the same order
## as in community data.

## Warning: Threshold reached by Scenario 1 alone

## $Original
## $Original$com_available
##      sp 1 sp 3 sp 5
## [1,]   50    0    0
## [2,]   40   10    0
## [3,]   30    0    5
##
## $Original$available_pool
## [1] 0.675
##
## $Original$available_plot
## [1] 0.6666667 0.8333333 0.5384615
##
##
## $Scenario1
## $Scenario1$sample_species
## [1] "sp 2"
##
## $Scenario1$com_available
##      sp 1 sp 2 sp 3 sp 4 sp 5
## [1,]   50   20    0   NA    0
## [2,]   40    0   10   NA    0
## [3,]   30   30    0   NA    5
##
## $Scenario1$available_pool
## [1] 0.925
##
## $Scenario1$available_plot
## [1] 0.9333333 0.8333333 1.0000000
```

If `sequential` would be set to `FALSE`, the species to be sampled according to Scenario 2 is independent from Scenario 1. The species indicated to be measured by Scenario 1 is the same as when sequential is set to TRUE, but Scenario 2 indicates species without taking Scenario 1 into account. Hence, the function now provides a full set of species that need to be sampled in each individual plot of the study. This would be an appropriate sampling strategy when for instance one would want to have plot-wise trait measurements, i.e. one could than take into account trait variation on the within-plot level.

It is important to note that these functions need some sort of (semi)quantitative abundance data (e.g. frequency, cover values, individual counts, biomass) and do not work with pure presence-absence community matrices.

Below we give an example of the `sampleSpecies` function with some real data from a meadow plant community in the Czech Republic. Originally, this data set contains 61 species in 12 plots, and accompanying trait data for the species. The community data and trait data are stored in the elements of a list object called `ohrazeni`, after the name of the location of the meadow. This data is part of the traitor package, so we need to first load it with the data function. For the purpose of demonstration, we cut the original 60 species down to 10 species, so that our R console (and this document) does not overflow with content.

```
ohrazeni$vegetationdata <- ohrazeni$vegetationdata[, 1:10]
ohrazeni$traitdata <- ohrazeni$traitdata[1:10, ]
ohrazeni
```

```
## $vegetationdata
##       sp1    sp2   sp3   sp4   sp5   sp6    sp7   sp8   sp9  sp10
## 1   0.010  0.000 3.863 0.000 0.868 2.103  0.000 3.135 0.021 0.029
## 2  27.675  0.016 0.665 0.232 1.387 0.068  5.928 0.469 0.012 0.000
## 3   0.240  0.000 0.136 0.000 0.377 1.113  0.000 1.492 0.000 0.000
## 4  77.943  0.000 0.129 0.000 0.305 0.862  0.071 0.249 0.000 0.000
## 5   0.347 28.678 0.992 0.410 1.611 0.002  0.000 0.000 0.000 0.000
## 6   0.019  1.245 1.639 0.000 0.537 1.130  0.000 0.727 0.000 0.000
## 7  24.808 10.784 0.000 0.000 0.048 1.360  0.173 0.039 0.000 0.000
## 8   0.000  3.035 5.603 0.000 0.130 3.326  0.000 3.573 0.055 0.000
## 9   0.000 27.886 0.000 0.000 0.000 0.000 57.711 0.000 0.000 0.000
## 10  2.185  0.000 0.080 0.000 0.184 1.185  1.035 0.838 0.000 0.000
## 11  0.933  5.020 0.000 0.000 0.086 0.970  0.000 1.401 0.000 0.000
## 12 11.282  0.000 0.000 0.000 1.985 0.359  0.225 0.000 0.000 0.000
##
## $traitdata
##      lifeform growthform leafposition   sla  ldmc height seedweight
## sp1      hemi      grass      semiros 28.30 275.9   60.5        0.0
## sp2      hemi      grass      semiros 27.70 294.3   39.5        0.1
## sp3      hemi       forb        erosu 23.80 227.4  105.0        0.2
## sp4      hemi       forb      semiros 30.40 166.2   45.0        1.3
## sp5      hemi       forb      semiros 28.40 187.4   75.0        0.7
## sp6      hemi      grass      semiros 23.30 322.0   27.5        0.5
## sp7      hemi       forb      semiros 25.00 217.1   65.0        0.6
## sp8      hemi      grass      semiros 18.90 254.5   32.5        0.1
## sp9      hemi      sedge      semiros 17.93 365.5   25.0        0.7
## sp10     hemi       forb      semiros 26.10 161.5   90.0        0.6
```

In this rare case, there are no NAs in the trait data. To simulate some more realistic dataset we create some NAs for one of the traits by replacing existing values with NA. There are actually two ways to specify missing data, via NA or as shown above via a vector of 0 and 1, where 0 means no trait data is available. Here instead directly the vector of trait values is used, where species with missing trait data are defined by the NAs. If in both data sets (i.e. trait and community data) species names are given, they are automatically matched by these names. If they are not named as such, the species in the trait data and the community data are assumed to be exactly in the same order. If they are not, `sampleSpecies` will indicate the wrong species to sample. As you might have already observed in the above examples, the function will issue a warning reminding you about this, when no species names are given.

```
sla_with_NA <- ohrazeni$traitdata$sla
sla_with_NA[sample(1:10, 4)] <- NA
names(sla_with_NA) <- rownames(ohrazeni$traitdata)
sampleSpecies(ohrazeni$vegetationdata, sla_with_NA, thresh = 0.9, sequential = FALSE)
```

```
## $Original
## $Original$com_available
##       sp1   sp5   sp6    sp7   sp8   sp9
## 1   0.010 0.868 2.103  0.000 3.135 0.021
## 2  27.675 1.387 0.068  5.928 0.469 0.012
## 3   0.240 0.377 1.113  0.000 1.492 0.000
## 4  77.943 0.305 0.862  0.071 0.249 0.000
## 5   0.347 1.611 0.002  0.000 0.000 0.000
```

```
## 6   0.019 0.537 1.130  0.000 0.727 0.000
## 7  24.808 0.048 1.360  0.173 0.039 0.000
## 8   0.000 0.130 3.326  0.000 3.573 0.055
## 9   0.000 0.000 0.000 57.711 0.000 0.000
## 10  2.185 0.184 1.185  1.035 0.838 0.000
## 11  0.933 0.086 0.970  0.000 1.401 0.000
## 12 11.282 1.985 0.359  0.225 0.000 0.000
##
## $Original$available_pool
## [1] 0.7284301
##
## $Original$available_plot
##  [1] 0.61192542 0.97495336 0.95949970 0.99837856 0.06117353 0.45554087
##  [7] 0.71020101 0.45057881 0.67421755 0.98547303 0.40309156 1.00000000
##
##
## $Scenario1
## $Scenario1$sample_species
## [1] "sp2"
##
## $Scenario1$com_available
##       sp1    sp2 sp3 sp4   sp5   sp6    sp7   sp8   sp9 sp10
## 1   0.010  0.000  NA  NA 0.868 2.103  0.000 3.135 0.021   NA
## 2  27.675  0.016  NA  NA 1.387 0.068  5.928 0.469 0.012   NA
## 3   0.240  0.000  NA  NA 0.377 1.113  0.000 1.492 0.000   NA
## 4  77.943  0.000  NA  NA 0.305 0.862  0.071 0.249 0.000   NA
## 5   0.347 28.678  NA  NA 1.611 0.002  0.000 0.000 0.000   NA
## 6   0.019  1.245  NA  NA 0.537 1.130  0.000 0.727 0.000   NA
## 7  24.808 10.784  NA  NA 0.048 1.360  0.173 0.039 0.000   NA
## 8   0.000  3.035  NA  NA 0.130 3.326  0.000 3.573 0.055   NA
## 9   0.000 27.886  NA  NA 0.000 0.000 57.711 0.000 0.000   NA
## 10  2.185  0.000  NA  NA 0.184 1.185  1.035 0.838 0.000   NA
## 11  0.933  5.020  NA  NA 0.086 0.970  0.000 1.401 0.000   NA
## 12 11.282  0.000  NA  NA 1.985 0.359  0.225 0.000 0.000   NA
##
## $Scenario1$available_pool
## [1] 0.9586288
##
## $Scenario1$available_plot
##  [1] 0.6119254 0.9753923 0.9594997 0.9983786 0.9562422 0.6905796 1.0000000
##  [8] 0.6436204 1.0000000 0.9854730 1.0000000 1.0000000
##
##
## $Scenario2
## $Scenario2$sample_species
## $Scenario2$sample_species[[1]]
## [1] "sp3"
##
## $Scenario2$sample_species[[2]]
## [1] "none"
##
## $Scenario2$sample_species[[3]]
## [1] "none"
##
## $Scenario2$sample_species[[4]]
## [1] "none"
##
## $Scenario2$sample_species[[5]]
```

```
## [1] "sp2"
##
## $Scenario2$sample_species[[6]]
## [1] "sp2" "sp3"
##
## $Scenario2$sample_species[[7]]
## [1] "sp2"
##
## $Scenario2$sample_species[[8]]
## [1] "sp2" "sp3"
##
## $Scenario2$sample_species[[9]]
## [1] "sp2"
##
## $Scenario2$sample_species[[10]]
## [1] "none"
##
## $Scenario2$sample_species[[11]]
## [1] "sp2"
##
## $Scenario2$sample_species[[12]]
## [1] "none"
##
##
## $Scenario2$com_available
##       sp1    sp2   sp3 sp4   sp5   sp6    sp7   sp8   sp9 sp10
## 1   0.010     NA 3.863  NA 0.868 2.103  0.000 3.135 0.021   NA
## 2  27.675     NA    NA  NA 1.387 0.068  5.928 0.469 0.012   NA
## 3   0.240     NA    NA  NA 0.377 1.113  0.000 1.492 0.000   NA
## 4  77.943     NA    NA  NA 0.305 0.862  0.071 0.249 0.000   NA
## 5   0.347 28.678    NA  NA 1.611 0.002  0.000 0.000 0.000   NA
## 6   0.019  1.245 1.639  NA 0.537 1.130  0.000 0.727 0.000   NA
## 7  24.808 10.784    NA  NA 0.048 1.360  0.173 0.039 0.000   NA
## 8   0.000  3.035 5.603  NA 0.130 3.326  0.000 3.573 0.055   NA
## 9   0.000 27.886    NA  NA 0.000 0.000 57.711 0.000 0.000   NA
## 10  2.185     NA    NA  NA 0.184 1.185  1.035 0.838 0.000   NA
## 11  0.933  5.020    NA  NA 0.086 0.970  0.000 1.401 0.000   NA
## 12 11.282     NA    NA  NA 1.985 0.359  0.225 0.000 0.000   NA
##
## $Scenario2$available_pool
## [1] 0.9919257
##
## $Scenario2$available_plot
##         1         2         3         4         5         6         7         8
## 0.9971084 0.9749534 0.9594997 0.9983786 0.9562422 1.0000000 1.0000000 1.0000000
##         9        10        11        12
## 1.0000000 0.9854730 1.0000000 1.0000000
```

As demonstrated, `sampleSpecies` can really make your live easier as it helps to identify which species you should select to measure trait data for, in the case that your trait data is not complete enough. For the example data we used here, the task is fairly easy, as we only need to sample a single species (`sp8`) to reach our set threshold of 90 % total abundance in each of the plots, and the pool level. Note though that this result is somewhat biased because of the many species we removed from the community data at the start to have a better overview of the output in the R console. In real life, you will probably not get away with sampling so few species, to fill gaps in your trait data.

3 – Trait dissimilarity

In this exercise we will learn how to compute and interpret trait *dissimilarity between pairs of organisms*, following Chapter 3 and part of Chapter 6 in the reference textbook. All theoretical and mathematical issues beyond the indices described in this exercise can be found there. Most of the examples will be focused on estimating dissimilarity between pairs of species, but the same approach can be used to estimate dissimilarity between individuals (as we show in a funny example in section 3, and further on the R material Ch 6).

3.1 Data

We will work first with some invented data, following the examples provided in Chapter 3 of the reference textbook (specifically Fig.3.3 and Fig. 3.4). We will then use some students' traits, collected in one of our previous courses ("students_traits.txt"). Finally we will consider some field data ("spxtNESpain.txt") with fuzzy coding traits of around 130 species collected along a climatic gradient in NE Spain (de Bello et al., 2006).

```
#file paths need to be substituted by URLs
students <- read.delim(here::here("data","chapter3","students_traits.txt"), row.names = 1)
spxtNESpain <- read.delim(here::here("data","chapter3","spxtNESpain.txt"), row.names = 1)
```

3.2 Required packages and additional functions

We also need to load various packages in order to run the analyses. Please ensure that you have installed these packages in your R library before you load them.

```
library(FD)
library(NbClust)
library(gtools)
```

As final step we need to read in a function not yet included in any package, i.e. *trova* described in de Bello et al. (2013). The function will be described also for the R material relative to Chapter 6, on intraspecific trait variability, and here we use it for fuzzy coded traits.

```
source(here::here("data/chapter3/trova.r"))
```

3.3 Calculation of trait dissimilarity

Let's start by recreating the example in Fig.3.3 in the reference book. To do this let us create first some *species x traits* matrix by hand, just for simplicity. This will include 3 traits, a quantitative trait such as *body size* (t1), a qualitative trait transformed into a binary code, for example if species are *carnivorous* or not (t2; 1=yes, 0=no) and a categorical trait with multiple levels (i.e. a factor with more than 2 levels), species *color* in the example (t3) using fuzzy coding (i.e. more columns are used to characterize a trait; see the reference book). We explain in more detail below, a bit better, the reasoning behind such fuzzy coding.

```
bodysize <- c(10, 20, 30, 40, 50, NA, 70)
carnivory <- c(1, 1, 0, 1, 0,1, 0)
red <- c(1, 0, 0.5, 0, 0.2, 0, 1)
yellow <- c(0, 1, 0, 0, 0.3, 1, 0)
blue <- c(0, 0, 0.5,1, 0.5, 0, 0)
colors.fuzzy <- cbind(red, yellow, blue)
names(bodysize) <- paste("sp", 1:7, sep="")
names(carnivory) <- paste("sp", 1:7, sep="")
rownames(colors.fuzzy) <-  paste("sp", 1:7, sep="")
tall <- as.data.frame(cbind(bodysize, carnivory, colors.fuzzy))
tall
```

```
##     bodysize carnivory red yellow blue
## sp1       10         1 1.0    0.0  0.0
## sp2       20         1 0.0    1.0  0.0
## sp3       30         0 0.5    0.0  0.5
## sp4       40         1 0.0    0.0  1.0
## sp5       50         0 0.2    0.3  0.5
## sp6       NA         1 0.0    1.0  0.0
## sp7       70         0 1.0    0.0  0.0
```

Let's now start by computing dissimilarity by hand, without using any existing function. We will do this for each trait separately, considering first the binary trait, e.g. carnivorous vs. not carnivorous species (1 and 0 respectively in trait *carnivory* in Fig. 3.3). The dissimilarity between two species is, in this case, simply given by whether the two species share the same carnivory level or not. For example, species 1 and 2 are both carnivorous so that the dissimilarity between them is zero. This can be computed simply as the difference of the trait value of species 1 (which is 1) and species 2 (also 1), as $abs(1-1) = 0$ (notice that we express this as absolute difference because dissimilarities cannot be negative). On the contrary species 1 and 3 are completely different in this trait, one is carnivorous and the other is not. In that case $abs(1-0) = 1$, which implies that the two species are 100% different between them. Following this, we can compute dissimilarity for all pairs of species using only carnivory. This can be done simply by considering the function `dist`. For example:

```
dist(tall$carnivory)
```

```
##   1 2 3 4 5 6
## 2 0
## 3 1 1
## 4 0 0 1
## 5 1 1 0 1
## 6 0 0 1 0 1
## 7 1 1 0 1 0 1
```

For some functions we do need trait dissimilarity in such a "triangular" shaped data. For others (e.g. `mpd` in *picante*) we need to transform such a "triangular" object into a matrix, for example as:

```
as.matrix(dist(carnivory))
```

```
##     sp1 sp2 sp3 sp4 sp5 sp6 sp7
## sp1   0   0   1   0   1   0   1
## sp2   0   0   1   0   1   0   1
## sp3   1   1   0   1   0   1   0
## sp4   0   0   1   0   1   0   1
## sp5   1   1   0   1   0   1   0
```

```
## sp6   0   0   1   0   1   0   1
## sp7   1   1   0   1   0   1   0
```

Following the examples above we can see that the trait dissimilarity is usually an object composed by either a triangle or by two specular/symmetric triangular matrices (such as in Fig. 3.3). Each of the triangles provides a dissimilarity value for each pair of species in a given dataset, and the two triangles contain the same information (most R functions need only the "triangle" type of data, i.e. class 'dist', others will need a quadrat, i.e. class 'matrix'). Now notice that by coding the trait "carnivory" between 0 and 1 (i.e. no vs. yes), all the dissimilarities between species are constrained between 0 and 1, which quite conveniently indicates that species are either equal between them or completely different. It is important to notice that the diagonal of this object is, in most cases, assumed to be equal to zero. This is so because in most approaches the dissimilarity of one species with itself is considered to be equal to zero, meaning that a species is equal (i.e. not different) to itself. Ideally this could be improved, i.e. including dissimilarity between individuals within a species, but most algorithms do not consider this case.

Let us now consider a *quantitative trait, body size*. Intuitively, the dissimilarity between one student 1.80 m tall and another 1.70 m tall is 10 cm. Similarly, in the example of *body size* (following Fig. 3.3) the dissimilarity between species will be, to start with, simply the difference in their observed trait values. For example, between species 1 and 2 it will be $abs(10-20) = 10$; between species 1 and 3 $abs(10-30) = 20$, and so on. However, a question arises: are these values big or small? In other words: how can we compare these values of dissimilarity with results from another traits, for example on *carnivory* (which is expressed in relation to its maximum possible obtainable value, i.e. 1)? How can we compare this trait to another trait expressed in another unit, for example grams? To allow such comparisons all traits need to *scaled to similar units*. To do so, most often dissimilarities are expressed on a 0-1 scale, as with the case of "carnivory", i.e. indicating the extreme cases of no difference (0) and maximum difference (1). To do this for quantitative traits we need to define what is the maximum possible dissimilarity. This can be done, for example as

```
tall$bodysize / max(tall$bodysize, na.rm=T) #notice we have an NA to take into account
```

```
## [1] 0.1428571 0.2857143 0.4285714 0.5714286 0.7142857        NA 1.0000000
```

Other approaches use the function `scale` in which the mean (center) of a quantitative trait is computed and the differences from such a mean are computed. The values are then standardized by the standard deviation of the trait. Of course you can apply this only to quantitative traits. Specifically a "center and standardization" is done for each column in a species x trait matrix, meaning that the mean of trait values (across all species) is subtracted from each trait value; then, the mean value equals 0 and becomes the "center". Then, these centered values are rescaled by dividing by the standard deviation of the trait values. In our case it would be (both by hand and using the built-in function):

```
(tall$bodysize - mean(tall$bodysize, na.rm=T)) / sd(tall$bodysize, na.rm = T) #by hand
```

```
## [1] -1.2344268 -0.7715167 -0.3086067  0.1543033  0.6172134         NA  1.5430335
```

```
scale(tall$bodysize) #with the built-in function
```

```
##             [,1]
## [1,] -1.2344268
## [2,] -0.7715167
## [3,] -0.3086067
## [4,]  0.1543033
## [5,]  0.6172134
```

```
## [6,]         NA
## [7,]  1.5430335
## attr(,"scaled:center")
## [1] 36.66667
## attr(,"scaled:scale")
## [1] 21.60247
```

The two approaches described above (standardization between 0 and 1 vs. scaling) are generally comparable. The advantage of the first is that it can be used, as we will see below, more easily to compare quantitative and categorical traits.

One of the most commonly used approaches to compute trait dissimilarity, the Gower distance (Botta-Dukát, 2005), follows the first approach. It considers, for quantitative traits, the highest dissimilarity for a given trait in a dataset to standardize the traits. In our case the maximum difference in body sizes is given by the difference between the biggest and the smallest species, i.e. $abs(70 - 10) = 60$. Once we know the maximum dissimilarity, we can simply divide the dissimilarity between species, in their original scale, by such maximum dissimilarity value in the data set. For example the dissimilarity between species 1 and 2 would be $abs(10 - 20)/60 = 0.17$, and between species 1 and 3 $abs(10 - 30)/60 = 0.33$. By doing this, the maximum dissimilarity between species is now 1.

```
dist(tall$bodysize) / max(dist(tall$bodysize), na.rm = T) #Gower by hand
```

```
##           1         2         3         4         5         6
## 2 0.1666667
## 3 0.3333333 0.1666667
## 4 0.5000000 0.3333333 0.1666667
## 5 0.6666667 0.5000000 0.3333333 0.1666667
## 6        NA        NA        NA        NA        NA
## 7 1.0000000 0.8333333 0.6666667 0.5000000 0.3333333        NA
```

Notice that we have here one species with a missing value, so that the dissimilarity between this species (sp6) and the rest of species cannot be computed at least for this trait. This is a problem as most functions that we will use to compute functional diversity cannot handle NAs. We will see below that this will not prevent us from computing the dissimilarity based on all traits together, if we have sufficient information for other traits.

Notice that scaling the dissimilarity between 0 and 1 is not only recommended to compare quantitative traits with other type of traits, but it makes some interpretations of dissimilarity easier and the computation of alpha and beta functional diversity more robust (Chapter 5). On the 0-1 scale, the Gower distance for quantitative traits represents how relatively different are two species compared to the maximum possible dissimilarity value. For example, the dissimilarity between species 1 and 3 = 0.33 in terms of body size, implies that species 1 and species 3 are 33% different of the possible maximum dissimilarity (which is 60 cm). This idea has a great advantage, meaning that we can compare traits between them, independently of the unit (e.g. meters vs. grams) because the dissimilarity of each trait would be standardized to its maximum value. Similarly, we can compare quantitative traits to other type of traits (between species 1 and 2 it is 0.17 for body size 1 for carnivory). Eventually it is possible to combine these two dissimilarities. Using Gower dissimilarity, this is simply expressed as an average of the two dissimilarities, using the arithmetic mean [(0.17+1)/2=0.58] or the geometric mean (Euclidean distance; (Pavoine et al., 2009)) [sqrt(0.17^{2+1}2)=1.01]. For the Gower dissimilarity a useful built-in function, `gowdis`, is provided in the package FD and, by default it computes the arithmetic mean:

```
library(FD)
gowdis(tall[, c("bodysize", "carnivory")]) #combining 2 traits
```

```
##            sp1        sp2        sp3        sp4        sp5        sp6
## sp2 0.08333333
## sp3 0.66666667 0.58333333
## sp4 0.25000000 0.16666667 0.58333333
## sp5 0.83333333 0.75000000 0.16666667 0.58333333
## sp6 0.00000000 0.00000000 1.00000000 0.00000000 1.00000000
## sp7 1.00000000 0.91666667 0.33333333 0.75000000 0.16666667 1.00000000
```

By hand it would be:

```
(dist(tall$carnivory) + dist(tall$bodysize) /  max(dist(tall$bodysize), na.rm=T)) / 2
```

```
##            1          2          3          4          5          6
## 2 0.08333333
## 3 0.66666667 0.58333333
## 4 0.25000000 0.16666667 0.58333333
## 5 0.83333333 0.75000000 0.16666667 0.58333333
## 6          NA         NA         NA         NA         NA
## 7 1.00000000 0.91666667 0.33333333 0.75000000 0.16666667         NA
```

The only difference in these two approaches is that the `gowdis` function does not provide NAs. This is because instead of computing the mean dissimilarity across the two traits, for those species with NA, only the dissimilarity of traits without NAs are used (for sp6 only carnivory is used in `gowdis`). We will come back to this below and in other sections as well.

3.4 Some problems with the Gower distance

> Notice that, after reading the following sections on the problems with the Gower distance and the fuzzy coded traits, we present a new R function designed to solve this problem (The `gawdis` function)

It should be noticed that combining quantitative and qualitative traits can pose several problems, which are often not recognized in the literature. Imagine a case, summarized in the R script below, where we have 20 species and two traits, one quantitative trait (where values between 1 and 100 are randomly drawn) and a binary trait with randomly attributed 0 and 1 values. If we compute the dissimilarity for each trait and them combine these two dissimilarities with a simple average (or even with Euclidean distance, i.e. geometric mean), the resulting dissimilarity will be affected much more by the binary trait than by the quantitative. In other words, the contribution of the binary trait to the overall dissimilarity will be disproportional. This is because the dissimilarity values with a binary trait are more extreme, they exhibit a greater variance (Pavoine et al., 2009). Extreme dissimilarities are more common in the binary trait: in our example, all species belonging to a different carnivory level will have a dissimilarity of 1, whereas for body size only a pair of species (species 1 and species 7) have this extreme dissimilarity value. As a consequence, the combined dissimilarity will be much more correlated (Pearson R>0.85 over multiple runs) with the dissimilarity computed using the binary trait only, rather than with the quantitative trait only (Pearson R<0.5; notice that since the script below is based on random values, these results could change when you run the script).

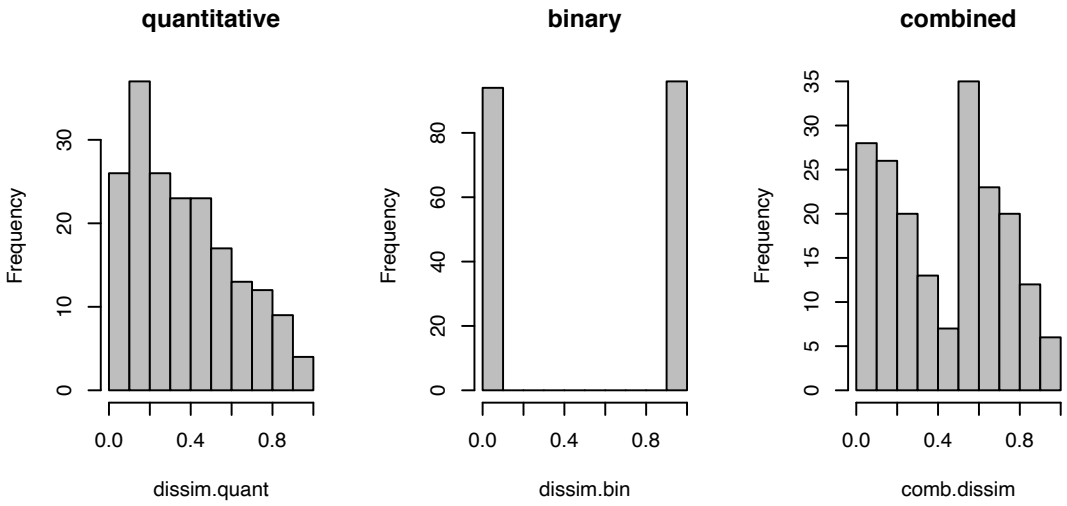
```
quantitative <- sample(1:100, 20)
binary <- sample(0:1, 20, replace = T)
dissim.quant <- dist(quantitative) / max(dist(quantitative))
dissim.bin <- dist(binary)
#comb.dissim<-(dissim.quant+dissim.bin)/2#gower distance by hand
comb.dissim <-gowdis(cbind(quantitative, binary)) #does the same with gowdis
cor(dissim.quant, comb.dissim)
```

```
## [1] 0.3983569
```

```
cor(dissim.bin, comb.dissim)
```

```
## [1] 0.8873016
```

```
par(mfrow = c(1, 3))
hist(dissim.quant, main = "quantitative", col = "grey")
hist(dissim.bin, main = "binary", col = "grey")
hist(comb.dissim, main = "combined", col = "grey")
```



One possible solution is to play with the *argument* `w` *in the function* `gowdis`. This argument allows us to give different weighs to the traits, i.e. when doing the average of dissimilarities across traits some traits could have more weight (i.e. a weighted average). For example, if we give more weight to the quantitative trait, for example 2 times the weight given to the binary traits, we get a more even importance of traits on the combined dissimilarity. In other words, by decreasing the weight of the categorical trait we avoid this trait to have a disproportional contribution to the multi-trait dissimilarity:

```
comb.dissim.weighted <- gowdis(cbind(quantitative, binary), w = c(2, 1))
cor(dissim.quant, comb.dissim.weighted)
```

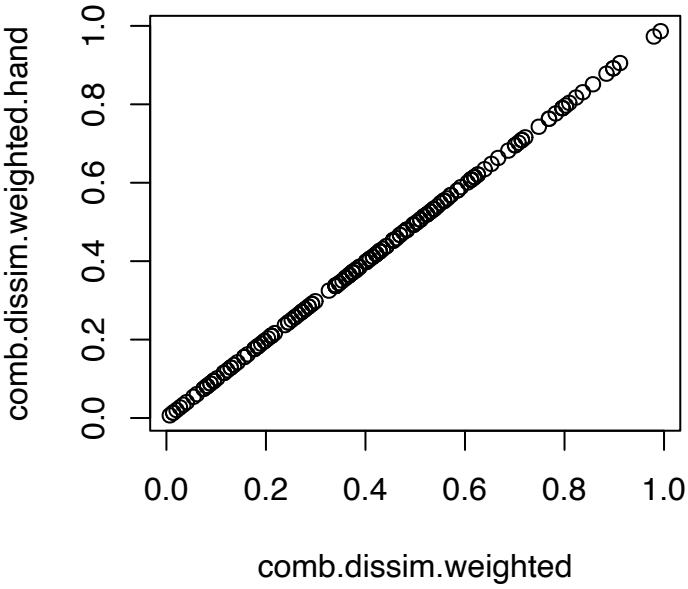
```
## [1] 0.6842637
```

```
cor(dissim.bin, comb.dissim.weighted)
```

```
## [1] 0.6798749
```

By hand, the weighted multi-trait dissimilarity would be computed as, for example:

```
comb.dissim.weighted.hand <-
  dist(quantitative / max(quantitative)) * 0.67 + dist(binary) * 0.33
plot(comb.dissim.weighted, comb.dissim.weighted.hand)
```



3.5 Categorical and fuzzy coded traits

Let us now consider the third trait we mentioned above and in Fig. 3.3 of the reference book, i.e. species color. How can we transform a trait with multiple, "unordered", levels into a quantitative trait? Is red bigger than blue? Ordering a trait such as this one is a difficult task (i.e. what is bigger and smaller?). Maybe in this case we could use the concentration of some pigment to define the color, resulting in only one column to define the trait. But in most occasions we have trait information expressed over multiple levels in which it is difficult to determine what is smaller or bigger. For example, herbaceous plant species could be grasses, legumes and fobs (a concept which was used to design the Jena experiment in Germany). This cannot be treated only as one value, as each of these strategies is considered to be completely different from another. Food source is another example, as usually there are multiple levels (italians eat spaghetti, frenchies croissant, czech dumplings, spanish tortilla and so on). For this type of traits, we could have two solutions. One is to code the trait as a factor. For example, imagine a case with 10 species, each being a grass, or a legume or a forb:

```
cat.trait <- sample(c("grass", "legume", "forb"), 10, replace = T)
df <- as.data.frame(cat.trait)
gowdis(as.data.frame(df))
```

```
##    1 2 3 4 5 6 7 8 9
## 2  1
## 3  1 0
## 4  0 1 1
## 5  0 1 1 0
## 6  1 1 1 1 1
## 7  0 1 1 0 0 1
## 8  0 1 1 0 0 1 0
## 9  1 0 0 1 1 1 1 1
## 10 1 1 1 1 1 0 1 1 1
```

This basically works as for the dissimilarity with a binary trait. If the species are into the same category, then dissimilarity is 0 and vice-versa, if the species are into different categories, then dissimilarity is 1. So we could treat this into a binary type of traits, but since we have three levels (grass, forb, legume) we would need 3 columns. This can be done, for example as:

```
acm.disjonctif(df)
```

```
##    cat.trait.forb cat.trait.grass cat.trait.legume
## 1               1               0                0
## 2               0               1                0
## 3               0               1                0
## 4               1               0                0
## 5               1               0                0
## 6               0               0                1
## 7               1               0                0
## 8               1               0                0
## 9               0               1                0
## 10              0               0                1
```

This function creates the so called *dummy variables*, where each level of a factor gets its own column. Notice that each species as a value 1 for one column and 0 for the others. So the sum across 3 columns, for a given species, is 1 (the `rowsum` will be equal to 1). **BE CAREFUL**, the function `gowdis`, does not need we perform this step by hand, so we gave give the trait information directly as the, for example, "df" object above. So that life is easy in this specific case.

On the other hand, the problem is when a species belongs simultaneously to different categories, for example in the (impossible) case that a species can be both a grass and a legume. But what if italians are not the only ones eating spaghetti and czech also eat spaghetti (even if with ketchup!)? The same can happen for the colors above, and for many categorical traits. What to do here? Ideally we can use the dummy variables with a *fuzzy coding* approach. For example, if the trait color has three main levels (red, blue, yellow, i.e. the three primary colors), then the trait can be summarized in the species x trait matrix by three columns, one for each color, called dummy variables. In the columns, we can then add 1 for a species if it belongs completely to one of the categories, out of the three in this case, in which the species belongs. For example, species 1 is red, so we will add 1 to the column referring to the red color and 0 to the other columns. For each row, the sum must equal to 1, because adding 1 into a column would reflect that the species is 100% having a given color (as species 1 is 100% red).

```
colors.fuzzy <- cbind(red, yellow, blue)
colors.fuzzy
```

```
##      red yellow blue
## [1,] 1.0    0.0  0.0
## [2,] 0.0    1.0  0.0
## [3,] 0.5    0.0  0.5
## [4,] 0.0    0.0  1.0
## [5,] 0.2    0.3  0.5
## [6,] 0.0    1.0  0.0
## [7,] 1.0    0.0  0.0
```

```
rowSums(colors.fuzzy)
```

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

Using dummy variables, combined with fuzzy coding, makes it possible to account for some species having an intermediate value among the different levels of the traits (i.e. having values that 'sit' between the different predefined categories of the trait). For example, a species of orange color could be codified as half red and half yellow. We could then add 0.5 to the red column and 0.5 to the yellow one (i.e. the species is 50% red and 50% yellow, which produces

orange). If the species is violet then it would be half blue and half red, as species 3 in the example above. These intermediate cases are actually quite frequent. This fuzzy coding approach can be also used for binary traits, for example some plant species are sometimes annual, sometimes perennial (or in the case of carnivory mentioned above, a species could be sometimes carnivorous and sometimes not, sort of part-time vegetarian). This can be solved by using one binary trait going from 0 to 1 but also considering intermediate values, like 0.5 for species that are equally in one form or the other. Imagine now a hypothetical species which is sometimes carnivorous and sometimes not, and assume we would code this species as 0.5 for the trait "carnivory". Then, the dissimilarity of this species with purely carnivore or not carnivore species (either having 0 or 1 in carnivory), would be $abs(1-0.5)$ or $abs(0-0.5)$, i.e. 0.5. Such a potential value of 0.5 would imply that species are 50% dissimilar.

Let us now see how the dissimilarity should be computed when using dummy variables, with either fuzzy coding or not. First let us assume that yellow, red and blue are completely different colors, for simplicity. This is an assumption we follow implicitly by using dummy variables. Then dissimilarity between species 1 and 2 should be equal to 1 (see object *colors.fuzzy* above), because each species "belongs" to different categories of the trait (they are included as 1 into the red and into the yellow columns respectively, i.e. they are 100% red and 100% yellow, respectively). On the other hand, the distance between species 1 and species 3 should be 0.5 because species 3 is partially red (50%). Similarly, the dissimilarity between species 1 and species 5 it should be 0.8, because species 1 and 5 overlap only in 20% of the cases.

There are several possible ways to compute this type of distances in R, but this should be done *very carefully*. Most important, most users are not aware that applying the commonly used function *gowdis* to the *species x trait* called *colors.fuzzy* will not work well, as it will not return these expected dissimilarity values. The problem is that the function will not "understand" that the three columns referring to species color are actually reflecting the same trait information. So it will consider that each column as a trait and will compute an average of the dissimilarity across all columns in the species x trait matrix:

```
gowdis(colors.fuzzy)
```

```
##           1         2         3         4         5         6
## 2 0.6666667
## 3 0.3333333 0.6666667
## 4 0.6666667 0.6666667 0.3333333
## 5 0.5333333 0.4666667 0.2000000 0.3333333
## 6 0.6666667 0.0000000 0.6666667 0.6666667 0.4666667
## 7 0.0000000 0.6666667 0.3333333 0.6666667 0.5333333 0.6666667
```

How to solve this? try this simple correction.

```
gowdis(colors.fuzzy) / max(gowdis(colors.fuzzy))
```

```
##     1   2   3   4   5   6
## 2 1.0
## 3 0.5 1.0
## 4 1.0 1.0 0.5
## 5 0.8 0.7 0.3 0.5
## 6 1.0 0.0 1.0 1.0 0.7
## 7 0.0 1.0 0.5 1.0 0.8 1.0
```

You can see that, now, the maximum dissimilarity is 1, and it is precisely between species that do not share any value in a given column. This is what we want!

The main problem appears if we want to compute dissimilarity using the object `tall`, created above, which includes the 3 traits created above (body size, carnivory and the 3 colors):

```
tall
```

```
##     bodysize carnivory red yellow blue
## sp1       10         1 1.0    0.0  0.0
## sp2       20         1 0.0    1.0  0.0
## sp3       30         0 0.5    0.0  0.5
## sp4       40         1 0.0    0.0  1.0
## sp5       50         0 0.2    0.3  0.5
## sp6       NA         1 0.0    1.0  0.0
## sp7       70         0 1.0    0.0  0.0
```

When combining the 3 traits to compute a combined dissimilarity, the gowdis function will actually "think" that there are 5 traits. When doing the average across the dissimilarities for the single trait it will thus give 1/5 of the weight to each column. By doing this it will wrongly give a weight of 1/5 to the quantitative trait, while it should be 1/3 and give the trait color a weight of 3/5 instead of 1/3. A temptation would be to use the argument `w` in the function gowdis, but this would not solve the problem. Unfortunately, in this case it is not simply possible to use the option of changing manually the weight of the columns, or at least we are not aware of a clear method that would do this correctly.

Instead, in such cases you need to compute dissimilarities for single traits individually and then average these dissimilarities across the different traits. Notice that `gowdis` does not work on vectors, but needs a matrix, or data frame, as the object containing the traits:

```
dissim.bodysize <- gowdis(as.matrix(bodysize))
dissim.carnivory <- gowdis(as.matrix(carnivory))
dissim.colour <- gowdis(colors.fuzzy) / max(gowdis(colors.fuzzy))
dall <-list(as.matrix(dissim.bodysize), as.matrix(dissim.carnivory),
    as.matrix(dissim.colour))
mean.dissim.all <- as.dist(apply(simplify2array(dall), c(1, 2),
                                 mean, na.rm = T), 2)
mean.dissim.all
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6       sp7
## sp1 0.0000000
## sp2 0.3888889 0.0000000
## sp3 0.6111111 0.7222222 0.0000000
## sp4 0.5000000 0.4444444 0.5555556 0.0000000
## sp5 0.8222222 0.7333333 0.2111111 0.5555556 0.0000000
## sp6 0.5000000 0.0000000 1.0000000 0.5000000 0.8500000 0.0000000
## sp7 0.6666667 0.9444444 0.3888889 0.8333333 0.3777778 1.0000000 0.0000000
```

or, which is the same (but more difficult if there are many traits and NAs):

```
(dissim.bodysize + dissim.carnivory + dissim.colour) / 3
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6
## sp2 0.3888889
## sp3 0.6111111 0.7222222
## sp4 0.5000000 0.4444444 0.5555556
## sp5 0.8222222 0.7333333 0.2111111 0.5555556
## sp6        NA        NA        NA        NA        NA
## sp7 0.6666667 0.9444444 0.3888889 0.8333333 0.3777778        NA
```

```
mean.dissim.all
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6       sp7
## sp1 0.0000000
## sp2 0.3888889 0.0000000
## sp3 0.6111111 0.7222222 0.0000000
## sp4 0.5000000 0.4444444 0.5555556 0.0000000
## sp5 0.8222222 0.7333333 0.2111111 0.5555556 0.0000000
## sp6 0.5000000 0.0000000 1.0000000 0.5000000 0.8500000 0.0000000
## sp7 0.6666667 0.9444444 0.3888889 0.8333333 0.3777778 1.0000000 0.0000000
```

we might also want to consider the function `dist.ktab` for this task (together with the associated functions `prep.fuzzy` and `ktab.list.df`). The function `dist.ktab` can combine different type of traits, including circular traits (like leaf angles, or flowering days etc). This is how it would work for the fuzzy trait colour created above:

```
colors.fuzzy2 <- prep.fuzzy(tall[, 3:5], col.blocks = 3)
#this apparently complicated step is just to make sure that the fuzzy traits are
#considered as such: indeed, the argument col.blocks defines how many columns
#characterize this trait
list.colour <- ktab.list.df(list(colors.fuzzy2))
dis.colour.ktab <- dist.ktab(list.colour, type = "F")
dis.colour.ktab
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6
## sp2 1.0000000
## sp3 0.5411961 1.0000000
## sp4 1.0000000 1.0000000 0.5411961
## sp5 0.8219228 0.7164745 0.4438974 0.4346181
## sp6 1.0000000 0.0000000 1.0000000 1.0000000 0.7164745
## sp7 0.0000000 1.0000000 0.5411961 1.0000000 0.8219228 1.0000000
```

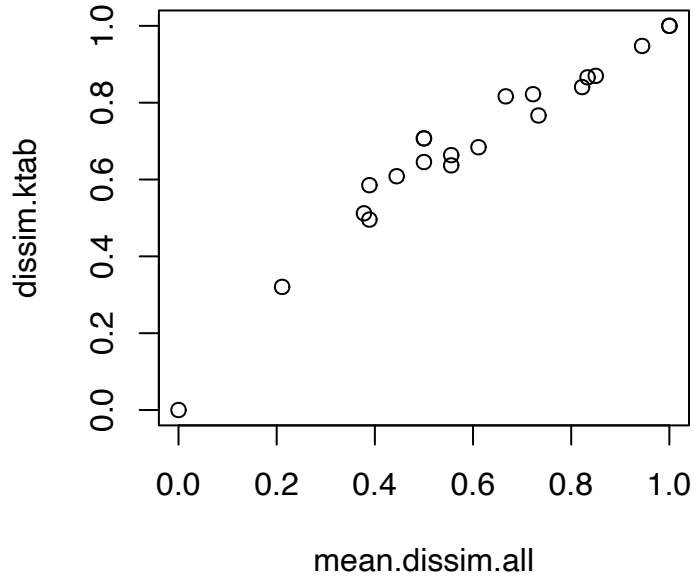
Now, when you look at these results you can notice that they do not correspond exactly to the object `dissim.colour` which we created above.

```
dissim.colour # the computation done above, by hand, for this trait
```

```
##     1   2   3   4   5   6
## 2 1.0
## 3 0.5 1.0
## 4 1.0 1.0 0.5
## 5 0.8 0.7 0.3 0.5
## 6 1.0 0.0 1.0 1.0 0.7
## 7 0.0 1.0 0.5 1.0 0.8 1.0
```

The differences are small, but not negligible. To the best of our understanding, which can be of course limited, we are not sure why `dis.colour.ktab` is not exactly as `dissim.colour` which, we understand, should be the preferred result. If we accept that the function `dist.ktab` provides the intended results (which we are not sure of) then we can also combine the trait colour with the other two traits in `tall` as:

```
list.traits <- ktab.list.df(list(tall[, 1:2], colors.fuzzy2))
dissim.ktab <- dist.ktab(list.traits,  c("Q", "F"))
plot(mean.dissim.all, dissim.ktab)
```



This shows that the results are different from what we hoped for. Alternatively, you can apply the function `trova` (de Bello et al., 2013) which can deal with this type of traits, and also circular traits and intraspecific trait variability. We will work with this function, and see how to use it below and in the R material connected to Chapter 6. See also R material 3: gawdis for a function which solves the problem with uneven trait contribution and can also handle fuzzy coded traits.

3.6 Missing values (NAs)

As already shown above, an important issue with the computation of trait dissimilarity is the case of *missing trait values*. In the case above of the matrix `tall`, for the quantitative trait the dissimilarity between species 6 and the other species cannot be computed (i.e. NA). The good news is that dissimilarity can be still computed considering other traits when, and only when, there is at least one trait value available for species 6 (and the species to which species 6 is compared to). The resulting dissimilarity combining traits for species 6 is simply the average of the dissimilarity for those traits without NA. In case of the example above this would correspond to the average between dissimilarity for carnivory and for color only, excluding body size (because of the NA). Then, we can still use the dissimilarity resulting from other traits and discard, for the comparisons with species 6, the missing information about body size. For example, the distance between species 1 and 6 would consider both color and carnivory, i.e. 0.5, since dissimilarity based on carnivory is equal to 0 and dissimilarity based on color is equal to 1, and so on. We will deal with the cases of filling NA values in another exercise related to Chapter 6.

3.7 Functional groups with students' traits

Let's now work with "real" data: the traits collected by students! Some students attended one of our courses, where we asked them to provide traits with the aim of splitting them into different "types". They provided various traits and here we use only 3 of them:

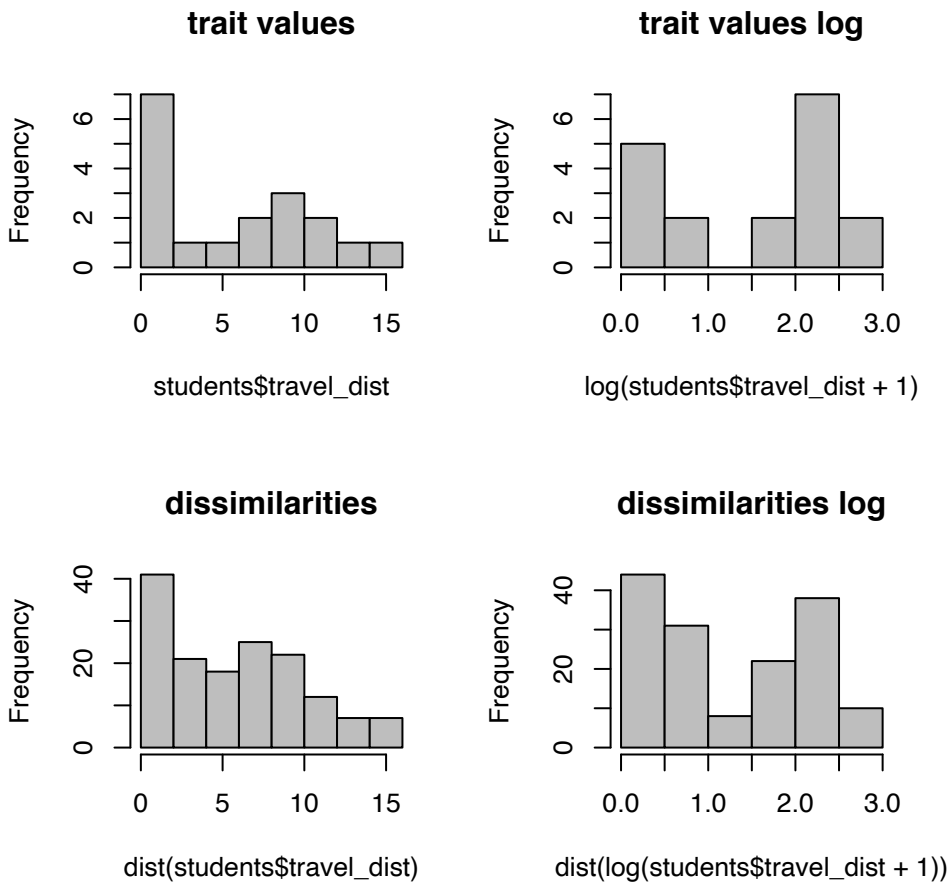
```
students #the data was uploaded at the beginning of this exercise
```

```
##          alc_cons km_bike_week travel_dist
## Ana             0            5        0.00
## Borbala         1            0        8.00
```

```
## Catarina        3           2        1.00
## Eduardo         6           0        0.00
## Elise           4          10       16.00
## Fernanda        5           0        0.00
## Giorgia         3          12        9.00
## Joana           1           0        0.00
## Johannes        6         120        9.00
## Julia           1          10        9.00
## Laura           7           2        4.00
## Hannah          0          15        7.00
## Martina         2           0        1.17
## Norbert         9         100       11.00
## Nuno           10           0        0.00
## Sanne           7          25        5.00
## Veronika        0           0       11.00
## Villem          4           5       14.00
```

In this case, the students had to say how much alcohol they consume (on a 1-10 scale, not specified here), how many kilometers by bike they do per week and how many hours they had to travel to come to the course. First, we can see that the last "trait" (of course this is surely not a functional trait) is not very uniformly distributed, because many students live very close to the university where the course took place. So those that come from very far will look like they are very different. However, if we log-transform the data we get a more "normal", uniform, distribution and the dissimilarities will be less skewed. Notice that, when we log-transform, we have to add "+1", as there are zero values in the trait.

```
par(mfrow = c(2, 2))
hist(students$travel_dist, col = "grey", main = "trait values")
hist(log(students$travel_dist + 1), col = "grey", main = "trait values log")
hist(dist(students$travel_dist), col = "grey", main = "dissimilarities")
hist(dist(log(students$travel_dist + 1)), col = "grey", main = "dissimilarities log")
```



So, as suggested by many authors in the case the trait is not normally distributed, we use the log-transformation, for example replacing the original values in the matrix as (although in some cases above the normality actually did not improve, but at least values did not look so wide spread after the transformation):

```
students$travel_dist <- log(students$travel_dist + 1)
```

We can also have a look at how the traits are correlated, just to see how much the single traits could be redundant

```
cor(students)
```

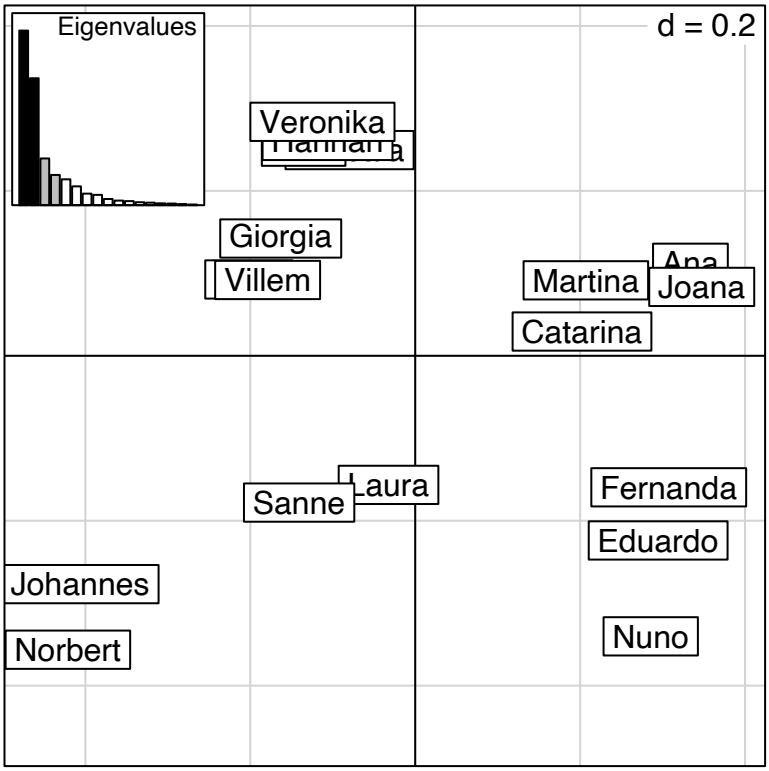
```
##                alc_cons km_bike_week travel_dist
## alc_cons      1.00000000    0.4029536 -0.07907633
## km_bike_week  0.40295355    1.0000000  0.38094912
## travel_dist  -0.07907633    0.3809491  1.00000000
```

In this case the correlations are relatively low. We can now compute the Gower dissimilarity between students, which in this case is simple, since there are no fuzzy coded traits:

```
distances.students <- gowdis(students)
```

The object produced is relatively big and difficult to visualize. How could we visualize, in a figure, how much are given students similar/dissimilar between them? We can use PCoA for this, since it relies on a dissimilarity matrix. Notice that in this case we could have even used PCA, because all traits were quantitative and there are no NA values, but often these conditions are not met. So let's run the PCoA. Sometimes, as in this case, using the square root of distance improves the characteristics of the dissimilarity and the function dudi.pco is "happier" (i.e. does not give warning messages):

```
pcoa.all <- dudi.pco(sqrt(distances.students), scannf = F, nf = 4)
scatter(pcoa.all)
```



How to read this? The students which are closer in the figure have more similar traits (for example Ana and Joana must be very similar between them from a functional point of view!). The first two axes of the PCoA are surely the most important ones (their eigenvalues are higher). We can compute how much of the variability is captured by the first two axes, which is around 65%.

```
sum(pcoa.all$eig[1:2]) / sum(pcoa.all$eig)
```

```
## [1] 0.6402358
```

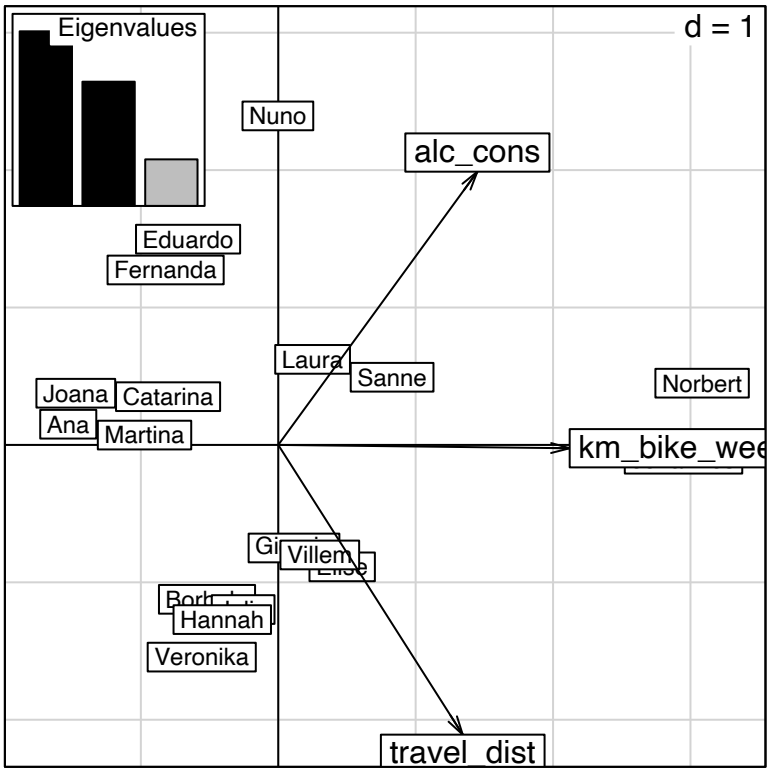
Are these axes related more to some traits than others? We can see if the position of students along a given axis (which is in the object "pcoa.all$li") is correlated to some traits. Let's compute the correlation between the position of the students on the first two PCA axes and the 3 traits

```
cor(pcoa.all$li[, 1:2], students)
```

```
##      alc_cons km_bike_week travel_dist
## A1 -0.1577509   -0.6710870  -0.9330768
## A2 -0.9287152   -0.4986334   0.3018606
```

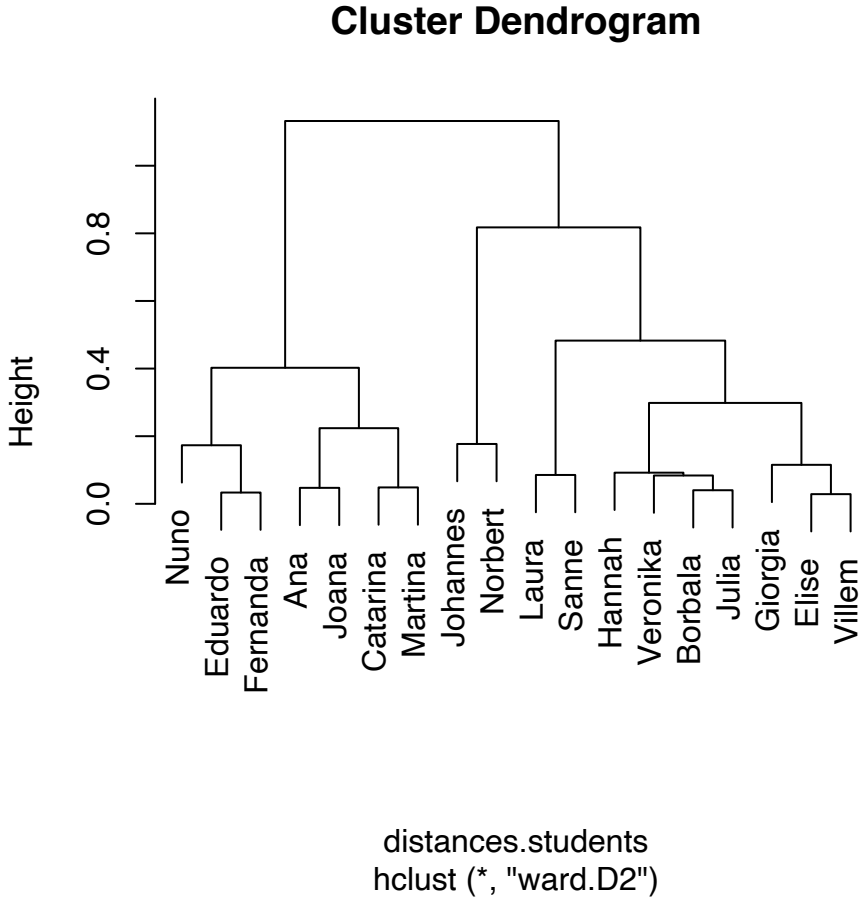
The first axis, the one accounting for most differences between students is generally representing their place of origins (people that travel few are on the left/upper side of the figure) and a bit how much they go by bike (who use the bike more are on the right). The second axis reflects alcohol consumption people on the above part of the graph drink less and also travel distance (those who travel more are on the lower part of the graph). Since we have only quantitative traits, without NA, we could also visualize the results with a PCA which provides, more or less, some comparable results:

```
scatter(dudi.pca(students, scannf = F, nf = 4))
```



We can now try to see if there are some different "types" of students, like trying to make "functional groups" of students based on the traits considered (see reference book, Chapter 3). This is easy to compute in R, but results will be generally quite variable, because there are many possible ways to do that. Anyway, one possible way could be the following one (see also material for Chapter 4). First, we can use one of the possible clustering options:

```
cluster.students <- hclust(distances.students, method = "ward.D2")
plot(cluster.students)
```



Then we can try one of many possible ways to find, statistically, the optimal number of groups in such a cluster (which maximizes the dissimilarity between groups and minimize the dissimilarity within groups). A good function is `NbClust` which offers a lot of possible approaches:

```
res.groups <- NbClust(diss = distances.students, distance = NULL, min.nc = 2,
                      max.nc = 6, method = "ward.D", index = "silhouette")
```

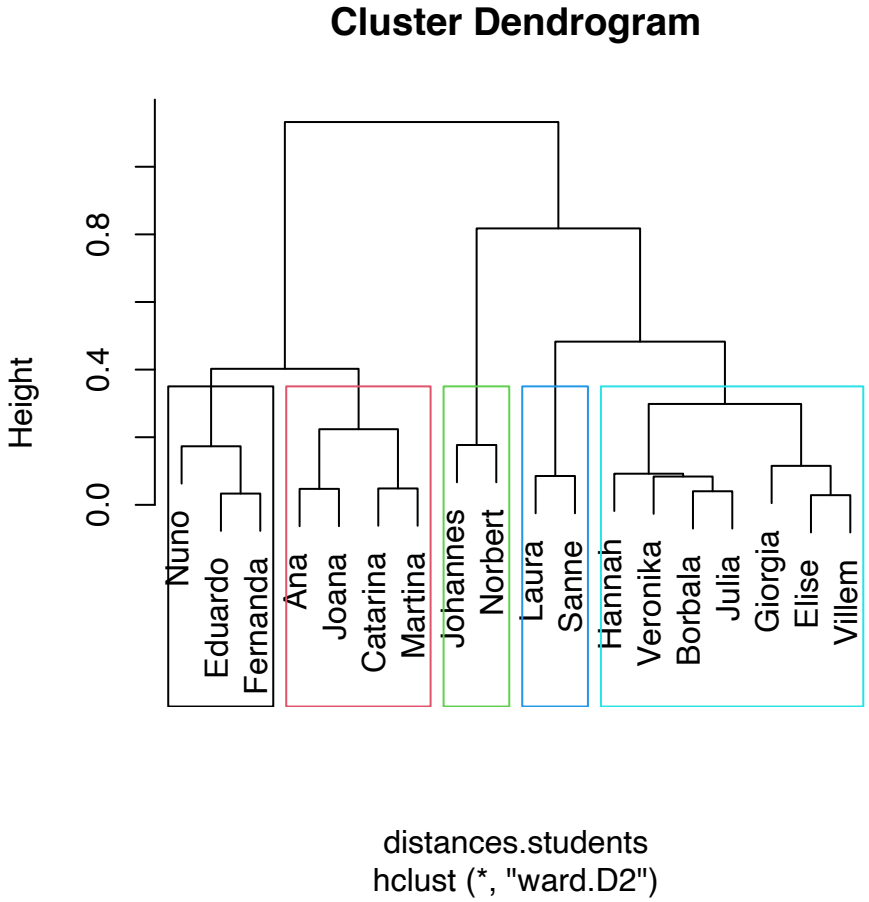
```
##
##  Only frey, mcclain, cindex, sihouette and dunn can be computed. To compute the other ind
```

```
res.groups
```

```
## $All.index
##      2      3      4      5      6
## 0.4309 0.5043 0.5105 0.5291 0.5282
##
## $Best.nc
## Number_clusters     Value_Index
##          5.0000          0.5291
##
## $Best.partition
##      Ana  Borbala Catarina  Eduardo    Elise Fernanda  Giorgia    Joana
##        1        2        1        3        2        3        2        1
## Johannes    Julia    Laura   Hannah  Martina  Norbert     Nuno    Sanne
##        4        2        5        2        1        4        3        5
## Veronika   Villem
##        2        2
```

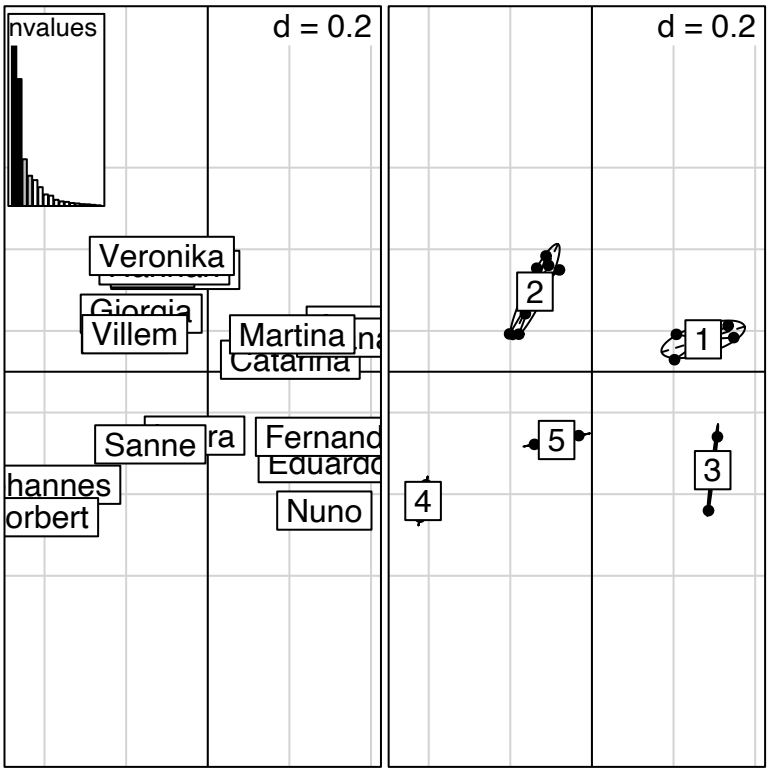
This tells us that, if we test how many groups (between 2 and 6) would maximize the difference between students, the answer is 5! We can visualize which students are in each group by exploring the object `$Best.partition` but also visually with a plot:

```
plot(cluster.students)
rect.hclust(cluster.students, k = res.groups$Best.nc[1],
            border = 1:res.groups$Best.nc[1])
```



we can also visualize the groups using `s.class`, with respect to the PCoA:

```
par(mfrow = c(1, 2))
scatter(pcoa.all)
s.class(pcoa.all$li, as.factor(res.groups$Best.partition), cpoint = 1)
```



3.8 Some real data with the function trova

Let's now focus on some field data, to show how the function trova works. The function allows to compute species dissimilarity not only using species means, but also considering overlap of trait values between species by using the standard deviation of a give trait. As explained in more detail in Chapter 6, this is done by computing the overlap in trait distribution for quantitative traits (see Lepš et al., 2006; de Bello et al., 2013). Also, it allows to use traits expressed in fuzzy coding as we saw above. The data is a subset of the data already published in several papers (de Bello et al., 2006, 2009). This data represent species sampled along climatic gradient in NE Spain. The gradient was characterized by *5 vegetation belts* going from a shrubland in Monegros, in the Aragon region (basically a desert with rainfall around 320 mm per year and altitude around 200 m a.s.l; Belt 1) to a subalpine meadow in the Catalan Pirinees (rainfall slightly below 1000 mm per year and altitude around 2000 m a.s.l; Belt 5). The traits of the species are in the file "spxtNESpain.txt" uploaded above, which includes information about 2 traits, SLA (specific leaf area) and Life Form. For SLA the data includes the mean SLA for each species and the standard deviation (SD). The life form ("LF_") is the Raunkiaer life form (Therophytes, Geophytes etc.). The file includes also an information of whether each species is present in the 5 vegetation belts considered. The five vegetation belt plots from the driest and warmer to wettest and coldest are characterized by a value from 1 to 5 (from 1=driest and warmest to 5=wettest and coldest). Let's have a look at the data:

```
head(spxtNESpain)
```

```
##           SLA  SD LF_Th LF_G LF_H LF_hCh LF_wCh LF_NP LF_P belt1 belt2 belt3
## Acercamp 15.7 2.4     0    0  0.0    0.0      0     0    1     0     0     1
## Achimill 14.6 1.6     0    0  0.5    0.5      0     0    0     0     0     0
## Aegigeni 15.2 2.6     1    0  0.0    0.0      0     0    0     0     1     1
## Alchhybr 19.0 1.7     0    0  1.0    0.0      0     0    0     0     0     0
## Anemhepa 12.9 0.5     0    0  1.0    0.0      0     0    0     0     0     1
## Anthmont 13.5 2.0     0    0  1.0    0.0      0     0    0     0     0     0
```

```
##          belt4 belt5
## Acercamp     0     0
## Achimill     1     1
## Aegigeni     0     0
## Alchhybr     0     1
## Anemhepa     1     1
## Anthmont     1     0
```

We will now extract only the trait data from this file, which is the first 9 columns:

```
spxt <- spxtNESpain[, 1:9]
```

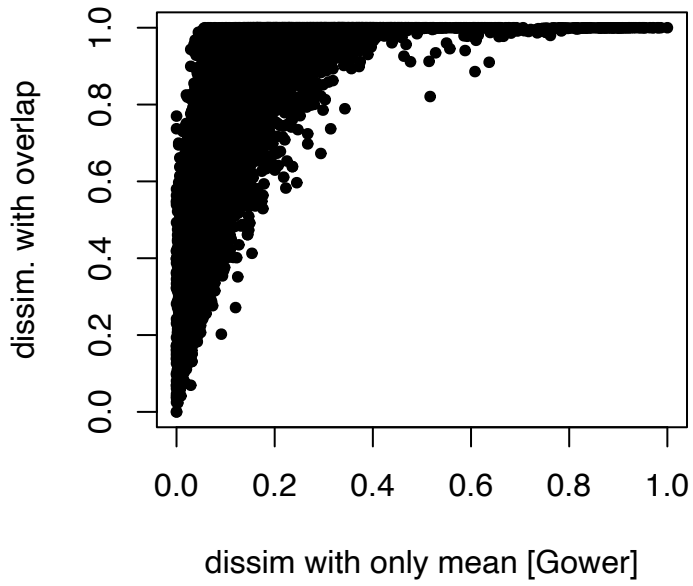
We can now use the `trova` function (uploaded above). This function needs us to give the name of species and introduce the different type of traits in different arguments. For those traits for which we have mean and SD we can use the so-called "gaussian.data". For the dummy variables and fuzzy coding we can use "multiple.category", where we need to provide a list (in which each object is a different trait; in this case we have only one dummy trait).

```
list.dummy <- list(spxt[, 3:9])
names(list.dummy) <- "LifeForm"
dissims <- trova(species.names = rownames(spxt), gaussian.data = spxt[, 1:2],
    multiple.category = list.dummy)
```

```
## [1] "Trait 1 / 2"
## [1] "Trait 2 / 2"
```

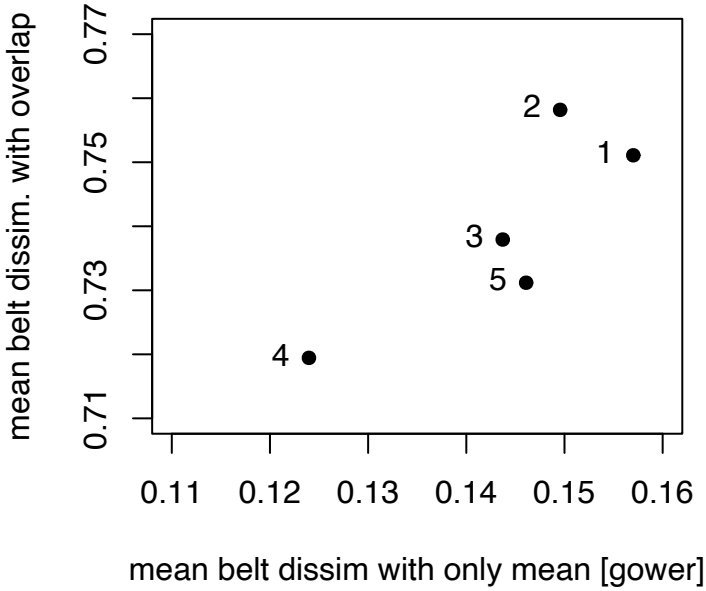
In this object we can extract the dissimilarity on each trait, as `dissims$SLA` or `dissims$LifeForm`. We would also be able to extract the average dissimilarity of these two traits with `dissims$gower`. We will now compare the results obtained with overlap (using mean and SD) with results from the gowdis function (i.e. using only species mean, without SD)

```
gowerSLA <- gowdis(as.data.frame(spxt$SLA))
plot(gowerSLA, dissims$SLA, pch = 20, ylab = "dissim. with overlap",
     xlab = "dissim with only mean [Gower]")
```



The figure shows all possible values of the dissimilarities, including all pair of species, so really a lot of points! it shows that with overlap the values are generally much bigger, although the two dissimilarities are quite correlated. If we would compare the average dissimilarity of species in the 5 vegetation belts, with the Gower approach vs. the overlap approach, we would get:

```
meangowerXbelt <- vector()
for (i in 1:5) {
  meangowerXbelt[i] <- mean(as.matrix(gowerSLA)[spxtNESpain[, i + 9] > 0,
                                                spxtNESpain[, i + 9] > 0])
}
meanoverlapXbelt <- vector()
for (i in 1:5) {
  meanoverlapXbelt[i] <- mean(as.matrix(dissims$SLA)[spxtNESpain[, i + 9] > 0,
                                                     spxtNESpain[, i + 9] > 0])
}
plot(meangowerXbelt, meanoverlapXbelt, pch = 20, cex = 1.3, xlim = c(0.11, 0.16),
  ylim = c(0.71, 0.77), ylab = "mean belt dissim. with overlap",
  xlab = "mean belt dissim with only mean [gower]")
text(meangowerXbelt, meanoverlapXbelt, 1:5, pos = 2)
```



This shows that the two approaches give some slight differences in results, for example higher dissimilarity to two different belts (the highest values was for belt 1 with Gower and was belt 2 with overlap), although we do not show here how to test if such ranking is significantly bigger/smaller, which could be done with some bootstrapping. Most important, it seems that the most stressed conditions (Belt 1 and 2 in the driest conditions) had higher dissimilarity between species. On the contrary, in Belt 4, maybe the least constrained, we had lower dissimilarity. Belt 5, although having greater rain had also greater altitude (~1900 m) and thus cold temperatures might also provide some stressful conditions. Please look at Chapter 7 for a deeper analyses on the effect of environmental conditions on species dissimilarity.

3.9 The `gawdis` function

For more information see the CRAN page for the `gawdis` package: https://cran.r-project.org/web/packages/gawdis/index.html

3.9.1 Introduction

In this step-by-step guide we will look how the function `gawdis` works and to apply it using simple data (either available in the FD package or made up below). The function `gawdis`, designed by Pavel Fibich, is an extension of the classic function `gowdis` in the package FD (Laliberté and Legendre, 2010) . The function computes dissimilarity between units, usually species, based on multiple types of variables (e.g. quantitative, categorical etc.), usually species' traits. Hence it can normally be applied to compute multi-trait dissimilarity between species in functional trait ecology studies, but it can be used in other applications as well. The dissimilarity obtained can be computed in order to attain a quasi-identical contribution of individual variables (e.g. traits) or group of associated variables (on variables reflecting similar information, e.g. multiple leaf traits). The dissimilarity is computed by either an analytical approach or through iterations. The function borrows several arguments from `gowdis`, with additional ones described below. This includes the option to consider fuzzy and dummy variables (e.g. multiple columns defining a single trait).

Let's first load functions.

3.9.2 Loading functions

We will now load the packages FD and gawdis, which you should have installed on your computer:

```
library(FD)
library(gawdis)
```

3.9.3 The Gower distance

We will start now by looking what the function does, overall. To do this let's first look at the original function `gawdis` with the data `dummy$trait`, which includes invented data for few species and their traits, available in the FD package.

```
dummy$trait
```

```
##     num1 num2 fac1 fac2 ord1 ord2 bin1 bin2
## sp1  9.0  4.5    A    X    3    2    0    1
## sp2  8.1  6.0    A    Z <NA>    1    0    1
## sp3   NA  2.3    C    Y    5    3    1    1
## sp4  3.2  5.4    B    Z    1    7    0    0
## sp5  5.8  1.2    C    X    2    6   NA    0
## sp6  3.4  8.5    C    Y    2    1    1    1
## sp7  7.5  2.1    B    X    3    2    1    0
## sp8  4.3  6.5 <NA>    Z    1    3    0    1
```

```
ex1 <- gowdis(dummy$trait)
round(ex1, 3)##just to see only 3 decimals, enough
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6   sp7
## sp2 0.218
## sp3 0.524 0.668
## sp4 0.674 0.561 0.823
## sp5 0.529 0.815 0.486 0.484
## sp6 0.610 0.593 0.278 0.707 0.607
## sp7 0.448 0.686 0.485 0.558 0.302 0.619
## sp8 0.407 0.204 0.596 0.239 0.559 0.447 0.703
```

```
class(ex1)
```

```
## [1] "dist"
```

The data `dummy$trait` include trait information for 8 species. Some traits are numerical (num1 and num2), some are categorical, i.e. factors (fac1 and fac2, nice names btw), some semi-quantitative traits, i.e. ordinal traits (ord1 and ord2) and some binary traits (bin1 and bin2). Some traits have NA values. The function `gawdis` computes the dissimilarity between the 8 species in the `dummy$trait`, based on all traits available. The function returns a 'triangular' distance object, of class `dist`, which express the (multi-traits) dissimilarity between each pair of species. Value are a scaled between 0 (species are exactly the same) and 1 (species are completely different).

Let's make even a simpler example, by taking only two traits (num2 and bin2) without NA. Let's see exactly what `gawdis` is doing. First a dissimilarity is computed for each trait. For the *quantitative trait* the differences in values between each pair of species are scaled to a maximum value 1, by dividing the dissimilarity values to the maximum possible difference for this trait. For example, when we looked at `dummy$trait` above, we would see that the sp6 had the highest value for num2 (i.e. 8.5) and sp7 had the lowest (i.e. 2.1). The difference between these two species will be equal to 1. For *binary trait* (e.g. if a species fly or not, 1 vs. 0), the maximum value is 1, so that there is no need to make such a standardization. Finally, the dissimilarity of the two traits is simply the *average of the distances for each individual trait*.

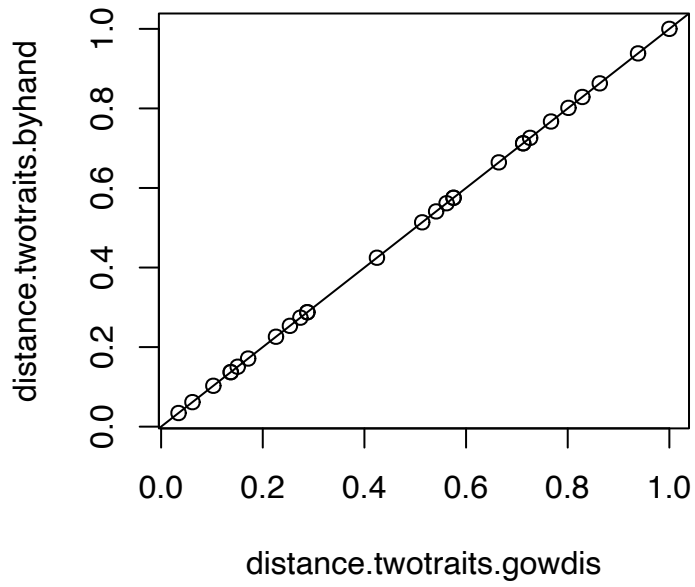
```
distance.num2 <- dist(dummy$trait[, "num2", drop = F]) / max(dist(dummy$trait$num2))
#note the drop=F not to loose the species names in the process#
round(distance.num2, 3)
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6   sp7
## sp2 0.205
## sp3 0.301 0.507
## sp4 0.123 0.082 0.425
## sp5 0.452 0.658 0.151 0.575
## sp6 0.548 0.342 0.849 0.425 1.000
## sp7 0.329 0.534 0.027 0.452 0.123 0.877
## sp8 0.274 0.068 0.575 0.151 0.726 0.274 0.603
```

```
distance.bin2 <- dist(dummy$trait$bin2)
distance.bin2
```

```
##   1 2 3 4 5 6 7
## 2 0
## 3 0 0
## 4 1 1 1
## 5 1 1 1 0
## 6 0 0 0 1 1
## 7 1 1 1 0 0 1
## 8 0 0 0 1 1 0 1
```

```
distance.twotraits.byhand <- (distance.num2 + distance.bin2) / 2
distance.twotraits.gowdis <- gowdis(dummy$trait[, c("num2", "bin2")])
plot(distance.twotraits.gowdis, distance.twotraits.byhand)
abline(0, 1)
```



These simple steps show what the function `gawdis` does "automatically" for us. Notice that, for the categorical trait the process is similar as for the binary traits. If the species belong to the same category, then the dissimilarity=0, if they belong to different categories, then the dissimilarity=0. For example:

```
dummy$trait$fac2
```

```
## [1] X Z Y Z X Y X Z
## Levels: X Y Z
```

```
gowdis(dummy$trait[, "fac2", drop = F]) #notice that gowdis wants a matrix, not a vector,
```

```
##     sp1 sp2 sp3 sp4 sp5 sp6 sp7
## sp2   1
## sp3   1   1
## sp4   1   0   1
## sp5   0   1   1   1
## sp6   1   1   0   1   1
## sp7   0   1   1   1   0   1
## sp8   1   0   1   0   1   1   1
```

```
#here a simple solution to solve this and keep species names
```

You can see that sp1 and sp2 have different categories (X and Z), so the dissimilarity is=1.

3.9.4 Traits contribution and trait weight

Let's now see why we actually need the new function `gawdis`, in addition to the traditional `gowdis`. In the example above `distance.twotraits.gowdis` reflects the dissimilarity of both traits, i.e. multi-trait dissimilarity, while `distance.num2` and `distance.bin2` reflect the dissimilarity for individual traits. What is the *contribution* of each single trait to the multi-trait dissimilarity? how much each single trait contribute to the final multi-trait dissimilarity? to answer this we can do, for example, a correlation between the multi-trait dissimilarity with the individual trait dissimilarity:

```
cor(distance.twotraits.gowdis, distance.num2)
```

```
## [1] 0.5167754
```

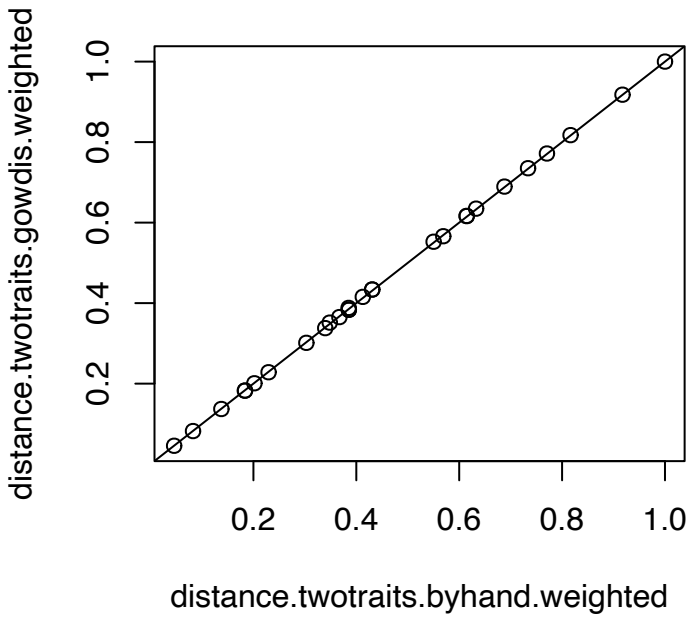
```
cor(distance.twotraits.gowdis, distance.bin2)
```

```
## [1] 0.8985725
```

This tells you how much the multi-trait dissimilarity reflects the information of each individual trait. If we look at this example we can see that the binary trait (bin2) is much more correlated to the multi-trait dissimilarity than the quantitative trait (num2), Pearson R=0.89 for the binary trait and R=0.51 for the quantitative. This means that the *contribution* of the binary trait is much much greater than of the quantitative trait. By the way, categorical/nominal traits will work similarly to binary traits, in general terms.

Are we happy with this result? are we happy that when we combine different variables (traits in this case), the dissimilarity has a far greater contribution from some variables? We think it is rather fair to say that this is not an ideal solution. Luckily the `gawdis` has an useful argument `w`, or *weight* which we can use to modify the contribution of each trait. Remember that the multi-trait dissimilarity is, by default with `gawdis`, a simple average of the dissimilarity from individual traits. Instead of doing a simple average, we could do a *weighted average*, in which some traits could have bigger weights. For example, we could reduce to the weight of the binary trait, to reduce its contribution to the multi-trait dissimilarity, and increase the one for the quantitative trait. In the next lines we show how this can be done "by hand" or we could be using directly the argument `w` in `gowdis` (notice that we are trying, to start with, to give twice the weight to the binary trait, compared to the quantitative one). See both options below:

```
distance.twotraits.byhand.weighted <- 0.67 * (distance.num2) + 0.33 * (distance.bin2)
distance.twotraits.gowdis.weighted <- gowdis(dummy$trait[, c("num2", "bin2")],
                                             w = c(2, 1))
plot(distance.twotraits.byhand.weighted, distance.twotraits.gowdis.weighted)
abline(0, 1)
```



By doing this the new multi-trait dissimilarity will be more evenly affected by each trait.

```
cor(distance.twotraits.gowdis.weighted, distance.num2)
```

```
## [1] 0.7454126
```

```
cor(distance.twotraits.gowdis.weighted, distance.bin2)
```

```
## [1] 0.7300753
```

Specifically the Pearson correlation is R=0.74 for the binary trait and R=0.73 for the quantitative. This is quite even, good job! What we did is modifying the weight of each trait to modify their contribution to the multi-trait dissimilarity. Cool! But…how do we know which values we have to select for the argument `w` to allow all traits to have a similar weight? if we have only 2 traits maybe we can try various `w` values and hope to find some decent combination, and try until we find a decent combination. The thing gets more complicated if we have many traits. Moreover, as we show below there are also other cases more complicated cases.

3.9.5 The function `gawdis`: basics

This is where the new function `gowdis` will get useful. The function looks for the best values for the `w` argument, to obtain an equal contribution of individual traits.

```
equalcont <- gawdis(dummy$trait[, c("num2", "bin2")])
```

```
## [1] "Running w.type=analytic"
```

```
equalcont
```

```
##            sp1        sp2        sp3        sp4        sp5        sp6
## sp2 0.13584757
## sp3 0.19924310 0.33509067
## sp4 0.42038371 0.39321419 0.61962681
## sp5 0.63773982 0.77358739 0.43849672 0.38037319
## sp6 0.36226018 0.22641261 0.56150328 0.61962681 1.00000000
## sp7 0.55623127 0.69207884 0.35698817 0.29886465 0.08150854 0.91849146
## sp8 0.18113009 0.04528252 0.38037319 0.43849672 0.81886991 0.18113009
##            sp7
## sp2
## sp3
## sp4
## sp5
## sp6
## sp7
## sp8 0.73736137
```

```
attr(equalcont, "correls")
```

```
##      num2      bin2
## 0.7377916 0.7377916
```

```
attr(equalcont, "weights")
```

```
##      num2      bin2
## 0.6611248 0.3388752
```

In one step we have a computed a multi-trait dissimilarity in which the contribution of each trait (provided in output of the function, i.e. in "correls") are basically identical. This was done by finding an ideal `w` value for each trait (provided in the output of the function, "weights"). Notice that the values are very close to the 2:1 ratio we 'tried' above with `distance.twotraits.byhand.weighted` and `distance.twotraits.gowdis.weighted`). Actually we 'tried' those values because we knew what was already the solution, but otherwise we would have spent some time really trying multiple combinations.

Of course the function `gawdis` can also be used in the exact same way as the original `gowdis`. For example, if you recall the object `ex1` created above with `gowdis`, and copied again below, now we can do the same with `gowdis`, just telling that we want to have a similar weight (and hence different contribution) for the different traits. This is the defaults in `gowdis` and in `gawdis` this is set by either using w.type ="equal" or by w.type ="user" and then saying `W` is the same for all traits.

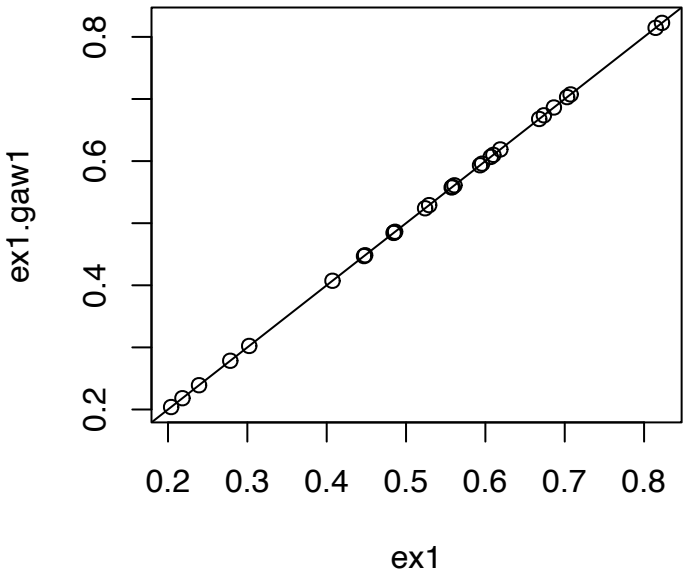
```
ex1 <- gowdis(dummy$trait)
ex1.gaw1 <- gawdis(dummy$trait, w.type = "equal")
```

```
## [1] "Running w.type=equal"
```

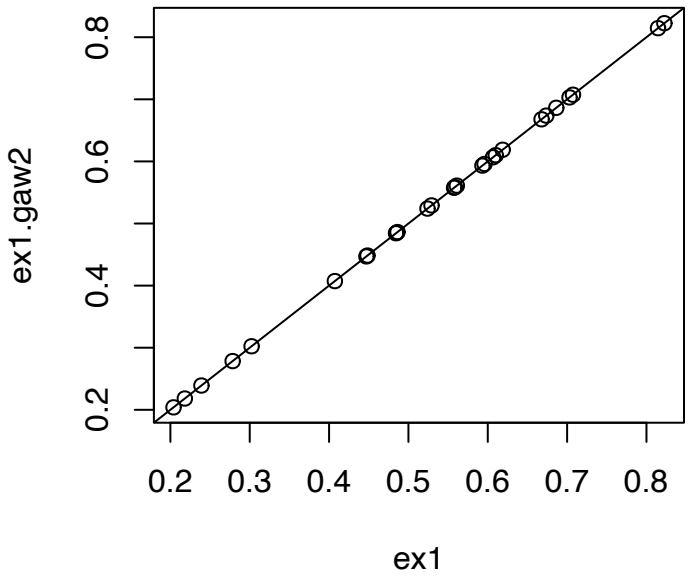
```
ex1.gaw2 <- gawdis(dummy$trait, w.type = "user", W = rep(1, 8))
```

```
## [1] "Running w.type=user"
```

```
plot(ex1, ex1.gaw1)
abline(0, 1)
```




```
plot(ex1, ex1.gaw2)
abline(0, 1)
```



3.9.6 The function `gawdis`: analytical vs. iterative approaches

Now that we verified that `gawdis` works fine, let's see more more details about the function and its different applications. The solutions provided by the function can be obtained in two ways. The first one is by an analytical solution, using purely a mathematical formulas (see main paper and corresponding supplementary material). This is the approach used by default in the function, as we used to create the object `equalcont` above. In case of doubts, this approach can be set by using w.type="analytic". NOTE that unfortunately this solution cannot be used when there are missing values (NAs).

When there are NAs, even if you set the argument w.type ="analytic" the function will apply a second approach, based on iterations, i.e. it will keep trying several solutions until finding a good one (actually it is based on a fitness improving approach, which gradually improves the output of the results). This will take some time, as multiple alternatives are tested sequentially. It can be obtained by using w.type ="optimized". The running time will depend, mostly, on the number of species and the number of iterations, which is set by the argument `opti.maxiter`, by default set to 300. Below we apply both approach to a subset of the `dummy$trait data`, without NAs

```
analytical <- gawdis(dummy$trait[, c(2, 4, 6, 8)], w.type = "analytic")
```

```
## [1] "Running w.type=analytic"
```

```
# it is not needed to add the argument w.type, this is the approach used by default if not defined
attr(analytical, "correls")
```

```
##      num2      fac2      ord2      bin2
## 0.5350139 0.5350139 0.5350139 0.5350139
```

```
attr(analytical, "weights")
```

```
##      num2      fac2      ord2      bin2
## 0.2422698 0.2467646 0.3575842 0.1533815
```

```
iterations <- gawdis(dummy$trait[, c(2, 4, 6, 8)], w.type = "optimized",
                     opti.maxiter = 100)
```

```
## [1] "Running w.type=optimized"
```

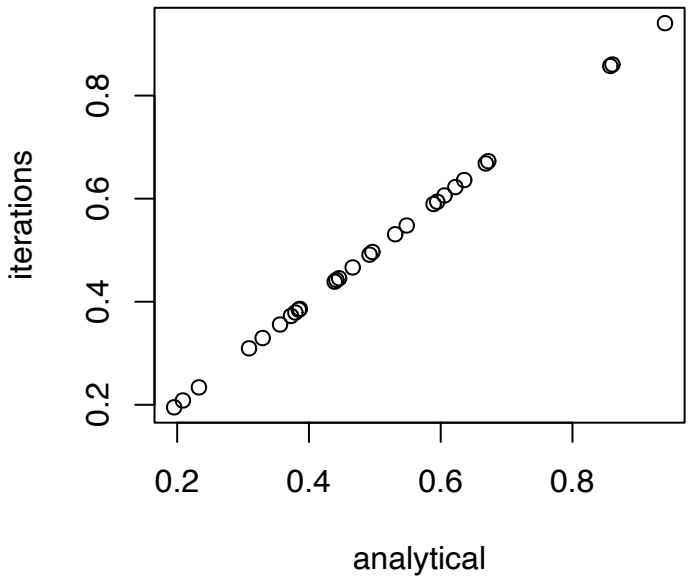
```
# here we used 'only' 100 iterations, to speed up the process and because with such few spec
attr(iterations, "correls")
```

```
##      num2      fac2      ord2      bin2
## 0.5353528 0.5343843 0.5343939 0.5364575
```

```
attr(iterations, "weights")
```

```
##      num2      fac2      ord2      bin2
## 0.2426619 0.2464132 0.3569084 0.1540165
```

```
plot(analytical, iterations)
```



Here you see some slight differences in the results, for example in terms of correlations between the dissimilarity of single traits and the multitrait (i.e. "correls") and the weights finally used for each trait ("weights"). The final results, the dissimilarity values displayed in the plot, also vary just a tiny bit. This is because the iterations are just based on a trial-error approach, which improves with more iterations, but it might "miss" the perfect mathematical solution, and just approach very close to it. Otherwise we can be quite confident that the results using the iterations tend quite well to the ideal results.

3.9.7 The function `gawdis`: grouping traits

In some cases the datasets we are considering have multiple variables which reflect some partially overlapping or redundant information. For example, many plant trait databases include a lot of leaf traits. This is the case, for example, of the dataset `tussock$trait`, also available in the FD package. Let's have a look:

```
dim(tussock$trait)
```

```
## [1] 53 16
```

```
head(tussock$trait)
```

```
##            growthform  height     LDMC leafN leafP leafS       SLA
## Achi_mill  Semi basal 0.09727 163.1264   3.4 0.310 0.210 15.637656
## Agro_capi  Semi basal 0.19100 302.7657   2.5 0.250 0.180 21.846627
## Anth_odor     Tussock 0.06350 280.2013   2.0 0.240 0.150 24.547661
## Apha_arve  Semi basal 0.04470 185.9606   1.7 0.290 0.180 18.587383
## Arre_elat     Tussock 0.56400 311.9551   2.6 0.256 0.174 20.777386
## Brac_bell Short basal 0.00338 223.5052   1.2 0.330 0.220  6.549107
##           nutrientuptake       raunkiaer          clonality  leafsize dispersal
## Achi_mill   Mycorrhizal      Chamaephyte Clonal belowground 313.51484      Wind
## Agro_capi   Mycorrhizal  Hemicryptophyte Clonal belowground 141.78019      Wind
## Anth_odor   Mycorrhizal  Hemicryptophyte               None 161.43904   Passive
## Apha_arve   Mycorrhizal       Therophyte               None  20.89516   Passive
## Arre_elat   Mycorrhizal  Hemicryptophyte Clonal belowground 805.42883   Passive
## Brac_bell   Mycorrhizal  Hemicryptophyte               None 216.33835      Wind
##           seedmass resprouting        pollination  lifespan
## Achi_mill   0.1600         Yes Hymenoptera (bees) Perennial
## Agro_capi   0.0580         Yes               Wind Perennial
## Anth_odor   0.5600         Yes               Wind Perennial
## Apha_arve   0.1590          No               Self    Annual
## Arre_elat   2.9000         Yes               Wind Perennial
## Brac_bell   0.0524         Yes Hymenoptera (bees) Perennial
```

```
head(tussock$trait[, 3:7])
```

```
##               LDMC leafN leafP leafS       SLA
## Achi_mill 163.1264   3.4 0.310 0.210 15.637656
## Agro_capi 302.7657   2.5 0.250 0.180 21.846627
## Anth_odor 280.2013   2.0 0.240 0.150 24.547661
## Apha_arve 185.9606   1.7 0.290 0.180 18.587383
## Arre_elat 311.9551   2.6 0.256 0.174 20.777386
## Brac_bell 223.5052   1.2 0.330 0.220  6.549107
```

```
cor(tussock$trait[, 3:7], use = "complete")
```

```
##             LDMC      leafN      leafP      leafS        SLA
## LDMC   1.0000000 -0.6848213 -0.6682552 -0.5366135 -0.6417381
## leafN -0.6848213  1.0000000  0.6052319  0.7546005  0.6384765
## leafP -0.6682552  0.6052319  1.0000000  0.7703387  0.6212733
## leafS -0.5366135  0.7546005  0.7703387  1.0000000  0.5966755
## SLA   -0.6417381  0.6384765  0.6212733  0.5966755  1.0000000
```

In the dataset there are 5 leaf traits, likely measured on the same leaves, and quite correlated between them. In this case we do suggest to create groups of traits, using the argument `groups`. Why it is so? the problem is that if many trait provide a similar information, actually all these leaf traits are very much correlated. So the information could become too prominent in the multi-trait dissimilarity. Let's see all this, with a step-by-step approach. Basically the same test can be found in the incoming publication (de Bello et al., 2021). First we slightly reduce the number of traits, just for simplicity, and because the dataset included some very unbalanced categorical traits (with too few entries for a number of categories). Then we need to log-transform some of the quantitative traits, and previously we checked that height, seed mass and leafS were the one needing log-transformation. NOTE that, by the way, that if we do not use log-transformation is applied trait with abnormal trait distribution will have bigger contribution, simply because they have greater variance (see Pavoine et al., 2009).

```
tussock.trait<-tussock$trait[, c("height", "LDMC", "leafN","leafS", "leafP",
                                 "SLA", "seedmass", "raunkiaer", "pollination",
                                 "clonality", "growthform")]
tussock.trait.log <- tussock.trait #some traits needed log-transformation,
#just creating a matrix to store the new data
tussock.trait.log$height <- log(tussock.trait$height)
tussock.trait.log$seedmass <- log(tussock.trait$seedmass)
tussock.trait.log$leafS <- log(tussock.trait$leafS)
colnames(tussock.trait.log)
```

```
##  [1] "height"      "LDMC"        "leafN"       "leafS"       "leafP"
##  [6] "SLA"         "seedmass"    "raunkiaer"   "pollination" "clonality"
## [11] "growthform"
```

Let's first compute the simple Gower distance with `gowdis`. We we will also compute the dissimilarity only on leaf traits and correlate it to the multi-trait dissimilarity:

```
#straightgowdis<-gowdis(tussock.trait.log)
straightgowdis.2 <- gawdis(tussock.trait.log, w.type = "equal", silent = T)
#we compute 'normal' gower with the new function because it provides more results
#plot(straightgowdis, straightgowdis.2)# if you want to check that the results are the same
cors.gow <- attr(straightgowdis.2, "correls")
cors.gow[12] <- vegan::mantel(straightgowdis.2, gowdis(tussock.trait.log[, 2:6]),
                              na.rm = T)$statistic
names(cors.gow)[12] <- "leaves"
cors.gow
```

```
##      height        LDMC       leafN       leafS       leafP         SLA
##   0.2023004   0.5083692   0.4522899   0.4944064   0.4292520   0.3729605
##    seedmass   raunkiaer pollination   clonality  growthform      leaves
##   0.2604944   0.5941643   0.4528619   0.3237053   0.4668307   0.5889437
```

We can see that the biggest contribution (across the `cors.gow` values) were obtained for a categorical traits, raunkiaer life form and the combination of leaf traits ('leaves'). This is simply because with w.type = "equal" the multi-trait dissimilarity is an average of the dissimilarity for single traits, so leaf traits are represented 5/11 times in this average, a disproportional effect with respect to other traits, right? We could say that, although there are differences between the leaf traits, they are the same TYPE of trait, counted 5 times, when combining the traits.

The problem is not solved by using PCoA analyses, as suggested by the current literature (with the idea to reduce the number of traits and synthesize correlated traits into some multivariate axis). Here is a clear example:

```
testpcoa <- dbFD(cailliez(straightgowdis.2), tussock$abun)
```

```
## FRic: Dimensionality reduction was required. The last 40 PCoA axes (out of 51 in total) w
## FRic: Quality of the reduced-space representation = 0.6098694
## CWM: When 'x' is a distance matrix, CWM cannot be calculated.
```

```
#checking how many PCoA axes are retained
pcoaaxes <- dudi.pco(cailliez(straightgowdis.2), scannf = FALSE, nf = 11)
gowdis.PCoA <- dist(pcoaaxes$li)
sum(pcoaaxes$eig[1:11]) / sum(pcoaaxes$eig) #how much variability the axes explain
```

```
## [1] 0.6098694
```

```
#contribution of traits on the combined dissimilarity, done by hand#
cors.pcoa <- vector()
for (i in 1:dim(tussock.trait.log)[2]) {
  cors.pcoa[i] <- vegan::mantel(gowdis.PCoA, gowdis(as.data.frame(tussock.trait.log[, i])),
                          na.rm = T)$statistic
}
cors.pcoa[12] <- vegan::mantel(gowdis.PCoA, gowdis(tussock.trait.log[, 2:6]),
                         na.rm = T)$statistic
names(cors.pcoa) <- c(colnames(tussock.trait.log), "leaves")  #contributions of
# traits to the overall multi-trait dissimilarity
cors.pcoa
```

```
##      height        LDMC       leafN       leafS       leafP         SLA
##   0.2167067   0.4735296   0.4298086   0.4453686   0.3826323   0.3335101
##    seedmass   raunkiaer pollination   clonality  growthform      leaves
##   0.2993358   0.5685074   0.4333025   0.3334111   0.4647386   0.5419322
```

Let's just, for simplicity, focus only on the final result of this part. If we look at the object `cors.pcoa` we can see very similar contribution of traits, as obtained with the previous approach `cors.gow`. So, in very simplified terms, the PCoA approach does not help very much to solve the case of many redundant/correlated traits, as the correlated traits still have, altogether, a superior contribution to the overall multi-trait dissimilarity.

The function `gawdis` could help, but only if we define groups. If we do not, let's see what happen:

```
gaw.nogroups <- gawdis(tussock.trait.log, w.type = "optimized",
         opti.maxiter = 300) #there are NAs so the iteration approach is the only possible
```

```
## [1] "Running w.type=optimized"
```

```
cors.gaw <- attr(gaw.nogroups, "correls")
cors.gaw[12] <- vegan::mantel(gaw.nogroups, gowdis(as.data.frame(tussock.trait.log[, 2:6])),
                        na.rm = T)$statistic
names(cors.gaw)[12] <- "leaves"
cors.gaw
```

```
##      height        LDMC       leafN       leafS       leafP         SLA
##   0.4020639   0.4172761   0.4124997   0.4122294   0.4038993   0.4090948
##    seedmass   raunkiaer pollination   clonality  growthform      leaves
##   0.4106530   0.4124672   0.4172263   0.4173321   0.4132644   0.5498634
```

Now, if we look at the `cors.gaw`, we can see the function, apparently, successfully accomplished its mission to obtain a quasi-equal contribution of each single trait. But, the solution provided this is not a good solution neither, because leaves, altogether, have a much bigger contribution that other traits (0.54 while other traits are around ~0.41). Again, the function considered each leaf trait separately, so that altogether leaf traits will have a greater contribution.

A better solution can be obtained by saying that all leaf traits belong to a given group. Thus `gawdis` will first compute a dissimilarity for all leaf traits together and then try to get for this leaf-combined dissimilarity the same weight as for other traits. Let's see it:

```
colnames(tussock.trait.log)
```

```
##  [1] "height"      "LDMC"        "leafN"       "leafS"       "leafP"
##  [6] "SLA"         "seedmass"    "raunkiaer"   "pollination" "clonality"
## [11] "growthform"
```

```
gaw.groups <- gawdis(tussock.trait.log, w.type = "optimized", opti.maxiter = 300,
                     groups.weight = T, groups = c(1, 2, 2, 2, 2, 2, 3, 4, 5, 6, 7))
```

```
## [1] "Running w.type=optimized on groups=c(1,2,2,2,2,2,3,4,5,6,7)"
## [1] "Traits inside the group were weighted - optimized."
```

```
#there are NAs so the iteration approach is the only possible
cors.gaw.gr <- attr(gaw.groups, "correls")
cors.gaw.gr[12] <- attr(gaw.groups, "group.correls")[2]
names(cors.gaw.gr)[12] <- "leaves"
cors.gaw.gr
```

```
##      height        LDMC       leafN       leafS       leafP         SLA
##   0.4407875   0.3557227   0.3445287   0.3302545   0.3000801   0.2990226
##    seedmass   raunkiaer pollination   clonality  growthform      leaves
##   0.4302656   0.4433350   0.4441559   0.4477918   0.4402655   0.4420553
```

Now if we look at the `cors.gaw.gr` we can see single leaf traits have lower contribution than other traits, but in combination (leaves), they have a comparable contribution! Of course it will be difficult to decide when and in which case group of traits should be defined. But we think that if traits are measured in the same organ and provide, to a good extent, some overlapping information, then they should be considered as a group.

3.9.8 The function `gawdis`: fuzzing coding and dummy variables

The function `gawdis` can be also useful in the case of traits coded as dummy variables or, as a specific case of this, as dummy variables. Let's first create this new trait matrix, `tall`.

```
bodysize <- c(10, 20, 30, 40, 50, NA, 70)
carnivory <- c(1, 1, 0, 1, 0, 1, 0)
red <- c(1, 0, 0.5, 0, 0.2, 0, 1)
yellow <- c(0, 1, 0, 0, 0.3, 1, 0)
blue <- c(0, 0, 0.5, 1, 0.5, 0, 0)
colors.fuzzy <- cbind(red, yellow, blue)
names(bodysize) <- paste("sp", 1:7, sep = "")
names(carnivory) <- paste("sp", 1:7, sep = "")
rownames(colors.fuzzy) <- paste("sp", 1:7, sep = "")
tall <- as.data.frame(cbind(bodysize, carnivory, colors.fuzzy))
tall
```

```
##     bodysize carnivory red yellow blue
## sp1       10         1 1.0    0.0  0.0
## sp2       20         1 0.0    1.0  0.0
## sp3       30         0 0.5    0.0  0.5
## sp4       40         1 0.0    0.0  1.0
## sp5       50         0 0.2    0.3  0.5
## sp6       NA         1 0.0    1.0  0.0
## sp7       70         0 1.0    0.0  0.0
```

The object `tall` includes 3 traits for 7 species, bodysize (quantitative), carnivory (binary/categorical) and colour. The last one is represented by 3 columns, one for each basic colour (yellow, red, blue). For example the first species (sp1), is red, so it gets 1 in the the 'red' column and 0 in the others. This type of variables, defined by multiple columns is called dummy variable, and in this specific case fuzzy coding, because the value in each column can be different from 0 and 1, with the sum over the 3 columns being equal to 1. For more information see (https://www.univie.ac.at/arctictraits/traitspublic_fuzzyCoding.php, https://stattrek.com/multiple-regression/dummy-variables.aspx).

We surely cannot apply `gowdis` to this type of matrices, because the function will "think" there is a total of 5 traits, since the matrix contains 5 columns, and will treat each of the 3 columns for colors as independent traits, resulting in a higher contribution of this single trait. Moreover the function gets a bit 'crazy' with this type of variables. Let's see why. We can use, to start with, the function only on the colors traits:

```
round(gowdis(tall[, 3:5]), 3)
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6
## sp2 0.667
## sp3 0.333 0.667
## sp4 0.667 0.667 0.333
## sp5 0.533 0.467 0.200 0.333
## sp6 0.667 0.000 0.667 0.667 0.467
## sp7 0.000 0.667 0.333 0.667 0.533 0.667
```

We can appreciate that these results are not correct. Species 1 (sp1) was red, Species 2 was yellow. If for simplicity we think that each column means a completely different colour, then we do expect the dissimilarity between these two species should be equal to 1. This was not the case. Similarly sp3 is half red and so the dissimilarity with sp1 should equal to 0.5. But this is not the case. What can we do to solve this? Do simply the following, i.e. divide the dissimilarity by the maximum dissimilarity value:

```
round(gowdis(tall[, 3:5]) / max(gowdis(tall[, 3:5])), 3)
```

```
##     sp1 sp2 sp3 sp4 sp5 sp6
## sp2 1.0
## sp3 0.5 1.0
## sp4 1.0 1.0 0.5
## sp5 0.8 0.7 0.3 0.5
## sp6 1.0 0.0 1.0 1.0 0.7
## sp7 0.0 1.0 0.5 1.0 0.8 1.0
```

So if we want to combine this trait (colour, defined by 3 columns in the `tall`), we need first to compute the dissimilarity for colour this way, and then combine it with the other traits, for example with an average. For example, if we want a simple average of the dissimilarity for the 3 traits:

```
dissim.bodysize <- gowdis(tall[, "bodysize", drop = F])
dissim.carnivory <- gowdis(tall[, "carnivory", drop = F])
dissim.colour <- gowdis(tall[, 3:5]) / max(gowdis(tall[, 3:5]))
dall <- list(as.matrix(dissim.bodysize), as.matrix(dissim.carnivory),
    as.matrix(dissim.colour))
mean.dissim.all <- as.dist(apply(simplify2array(dall), c(1, 2), mean,
                                 na.rm = T), 2)
round(mean.dissim.all, 3)
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6   sp7
## sp1 0.000
## sp2 0.389 0.000
## sp3 0.611 0.722 0.000
## sp4 0.500 0.444 0.556 0.000
## sp5 0.822 0.733 0.211 0.556 0.000
## sp6 0.500 0.000 1.000 0.500 0.850 0.000
## sp7 0.667 0.944 0.389 0.833 0.378 1.000 0.000
```

This is all a bit time consuming to do it line by line. There is other function in the package 'ade4', i.e. the function `dist.ktab` (together with the associated functions `prep.fuzzy` and `ktab.list.df`). However this solution is quite time consuming as well, as it requires a number steps and coding lines (you can surely try, just in case). Ideally we can thus also solve this problem with the function `gawdis`, possibly in a simple way. The solution is obtained by defining all columns belonging to a given trait (like colors above) to a certain group, as we saw above and then setting the argument fuzzy=TRUE. Let's see this now.

```
gawdis(tall, w.type = "equal", groups = c(1, 2, 3, 3, 3), fuzzy = TRUE)
```

```
## [1] "Running w.type=equal on groups=c(1,2,3,3,3)"
## [1] "Running w.type=equal on groups=c(1)"
## [1] "Running w.type=equal on groups=c(2)"
## [1] "Running w.type=equal on groups=c(3,3,3)"
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6
## sp2 0.2777778
## sp3 0.5555556 0.6111111
## sp4 0.3888889 0.3333333 0.5000000
## sp5 0.7333333 0.6555556 0.1777778 0.5000000
## sp6 0.3333333 0.0000000 0.8333333 0.3333333 0.7333333
## sp7 0.6666667 0.8333333 0.3333333 0.7222222 0.2888889 0.8333333
```

4 – (Multivariate) species level responses

As mentioned in Chapter 4 of the reference book, there are a number of methodological approaches that allow us to examine trait-environment relationships at the species level. With species level, we mean the case that in our data each species is a data point. Specifically, we examine whether species environmental preferences are related to species traits. In other words, we test whether the environmental conditions in which different species occur, or prefer to occur, is determined or reflected by their traits. (Note that we can also look at the variability within species, but we treat this specifically in R material Ch 6 on intraspecific trait variability.) In most cases, analyses at the species level will happen by means of multivariate analyses. The important and visually obvious distinction to community level analyses is that species are the (data)points in our graphical outputs.

In general, the methods presented in this section often take a community matrix and a matrix of environmental conditions for these communities to obtain for each species its position along the axes of the resulting ordination. These positions can be referred to as 'response' to the environment, or as 'niche', and can then be related to the species traits, for which we need a third matrix containing the trait data (species by traits).

While we do not demonstrate the whole set of available methods for these analyses (which are mostly reviewed in Kleyer et al., 2012), we provide an exemplary look at a couple of important ones. One main distinction between available methods is the sequence in which the three different matrices containing the data (i.e. species composition, trait data, environmental data) are analyzed and combined.

4.1 Data

For the first part of this R material, we will use a data set provided and used by Kleyer et al. (2012). We start by loading in the different data sets that we need for conducting the analyses.

```
com <- read.delim(file = here::here("data", "chapter4", "Site_species.txt"))
traits <- read.delim(file =  here::here("data", "chapter4", "Species_traits.txt"))
env <- read.delim(here::here("data", "chapter4", "Site_env.txt"))
```

The data are:

(a) the community data (i.e. the species by site matrix) These are 50 species and their abundances across 43 plots in a grassland that was grazed by various livestock.

```
head(com[ , 1:10])
```

```
##           ACHIMILL AGROCAPI ANTHODOR BRIZMEDI BROMHORD CAPSBURS CAREFLAC
## GE.MNP_1      10.0     11.0      0.0        0      0.9      1.2        0
## GE.MNP_10      0.0      0.6      0.0        0     45.0      2.4        0
## GE.MNP_11     24.0      0.3      0.3        0      0.0      0.0        0
## GE.MNP_12      0.7      1.3      0.7        0      0.0      0.0        0
## GE.MNP_13      0.0      0.0      0.0        0      0.0      0.0        0
## GE.MNP_14     14.0      8.5      0.0        0      2.1      0.0        0
##           CAREHIRT CARENIGR CENTJACE
## GE.MNP_1       0.0        0      0.0
```

```
## GE.MNP_10      0.0        0      0.0
## GE.MNP_11      1.0        0      2.4
## GE.MNP_12      2.6        0      1.3
## GE.MNP_13      2.3        0      0.0
## GE.MNP_14      0.0        0      0.0
```

(b) the trait data (i.e. a trait by species matrix) The traits considered where flowering mode (`Polycarpic`, binary coded), leaf C:N ratio (`Cnratio`), seed mass (log-transformed, `seed.mass.log`), specific leaf area (`SLA`), canopy height (`height`, in cm), onset of flowering date (in Julian days, `Onset.flowe`).

```
head(traits)
```

```
##          Polycarpic Cnratio seed.mass.log   SLA height Onset.flower
## ACHIMILL          1  12.940        -0.876 19.63 21.150          178
## AGROCAPI          1  17.421        -1.444 29.54 21.017          167
## ANTHODOR          1  13.778        -0.678 29.97 14.967          170
## BRIZMEDI          1  33.690        -0.578 25.93 20.167          163
## BROMHORD          0  16.183         0.305 26.19 12.367          160
## CAPSBURS          0  11.624        -0.996 24.94 15.267          198
```

(c) the environmental data (i.e. a site by environmental variable matrix). These comprised grazing intensity, soil water-holding capacity, and extractable phosphorus. These variables are abbreviated as `dist.int` (for disturbance intensity), `SOIL.WHC`, and `SOIL.P`, respectively.

```
head(env)
```

```
##           dist.int SOIL.P SOIL.WHC
## GE-MNP_1      47.2  0.077      6.0
## GE-MNP_10     49.7  0.054      6.1
## GE-MNP_11     44.1  0.018     23.2
## GE-MNP_12     43.9  0.019     33.1
## GE-MNP_13     44.9  0.013     26.8
## GE-MNP_14     48.5  0.015     19.5
```

This data stem from Garnier et al. (2007). See this paper for a detailed description of the study site and the sampling methods.

For the part of this R material that covers the topic of boosted regression trees (section 9), we will use additional data that comprises species data for a set of different variables. We will load this data now, and will have a look in more detail when we get to the respective section.

```
meadows <- read.table(here::here("data", "chapter4", "meadows.txt"), header = T, stringsAsFactors
```

4.2 Required packages and additional functions

We also need to load various packages in order to run the analyses. Please ensure that you have installed these packages in your R library before you load them.

```
library(ade4)
library(MASS)
library(vegan)
library(ecodist)
library(maptools)
library(rpart)
library(splines)
library(gam)
library(pgirmess)
library(utils)
library(combinat)
library(cluster)
library(fpc)
library(clusterSim)
library(lmtest)
library(Hmisc)
library(gplots)
library(NbClust)
library(rpart)
library(rpart.plot)
library(dismo)
library(multcomp)
library(gbm)
library(raster)
```

As a final step of preparation, we need to read in some additional functions that are not part of any published packages, and were provided in the Supplementary Materials to Kleyer et al. (2012).

```
source(here::here("data", "chapter4", "Inference_modelset.r"))
source(here::here("data", "chapter4", "Inference_compute.r"))
source(here::here("data", "chapter4", "corratio.R"))
source(here::here("data", "chapter4", "calinski.R"))
source(here::here("data", "chapter4", "VarScoreOMI.r"))
source(here::here("data", "chapter4", "doublerda.R"))
```

4.3 "Trait-free" Canonical Correspondance Analysis

For comparison, and to start with a less complex example, we will perform a "trait-free" canonical correspondence analyses (CCA) on the community and environmental data. This will demonstrate how the species are distributed along the different environmental gradients considered, without taking into account the species traits yet. This first step will allow us to determine species preferences along the considered gradients.

We use the `cca` function from the `vegan` package. Although in the R code of the `cca` function it is also possible to provide the environmental variable as a single matrix, it is better to specify each variable in the formula and then provide in which object these variables can be found with the `data` argument. In fact, this enables us to examine the significance of each variable separately, as we will see later. In the case that the environment is only described by one variable, there is no difference of course.

For the data used here then, instead of

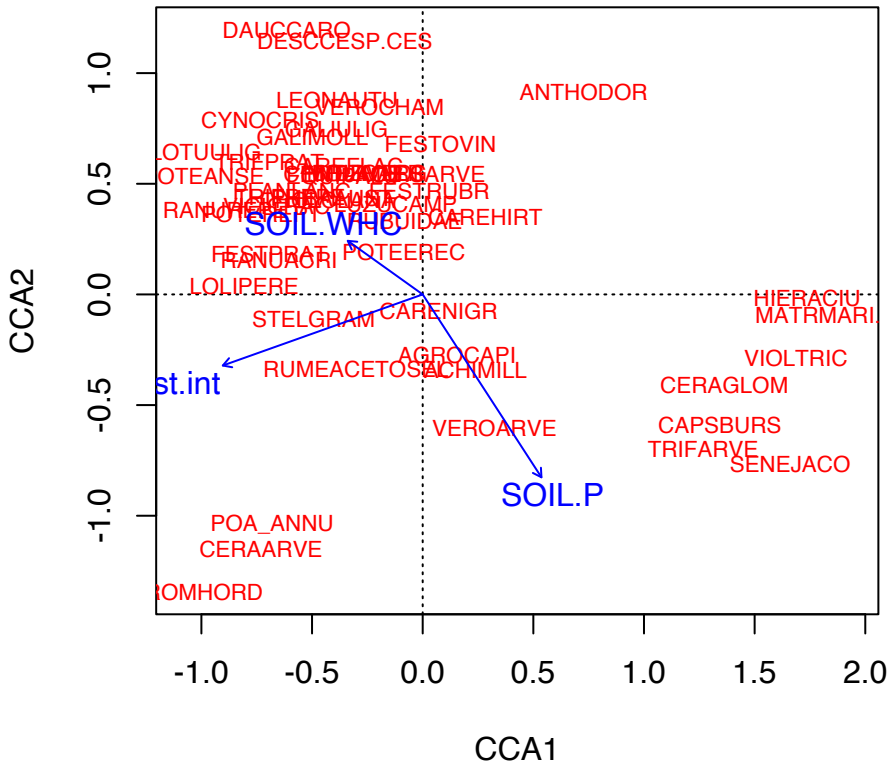
```
cca1 <- cca(com, env)
```

we run the CCA by

```
cca1 <- cca(com ~ dist.int + SOIL.P + SOIL.WHC, data = env)
```

First, we have a look at a simple plot that visualizes the CCA.

```
plot(cca1, display = c("species", "bp"))
```



In this graph, we see the species displayed by their names in red in their positions in the ordination. From this we can already see that certain species are located at the ends (i.e. towards the 'extremes') of certain gradients. For instance, SENEJACO (Senecio jacobaea - "Stinking Willie"), seems to prefer soils with high contents of available phosphor but not too wet, and sites with relatively low grazing pressure.

We can use a permutation based ANOVA to see which of the examined environmental variables significantly explain the ordination of the species. In addition, the overall explained variation in species compositions can be assessed by R^2, which is also computed using such a permutation approach.

```
anova(cca1, by = "terms")
```

```
## Permutation test for cca under reduced model
## Terms added sequentially (first to last)
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = com ~ dist.int + SOIL.P + SOIL.WHC, data = env)
##          Df ChiSquare      F Pr(>F)
## dist.int  1    0.4493 2.8797  0.001 ***
## SOIL.P    1    0.3833 2.4568  0.001 ***
## SOIL.WHC  1    0.2321 1.4876  0.044 *
## Residual 39    6.0847
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
RsquareAdj(cca1)
```

```
## $r.squared
## [1] 0.1489186
##
## $adj.r.squared
## [1] 0.0837634
```

The output from the ANOVA suggests that all environmental variables contribute to explain the species ordination, and from the adjusted R^2 we see that overall roughly 8% of variation in the species data is explained by these variables.

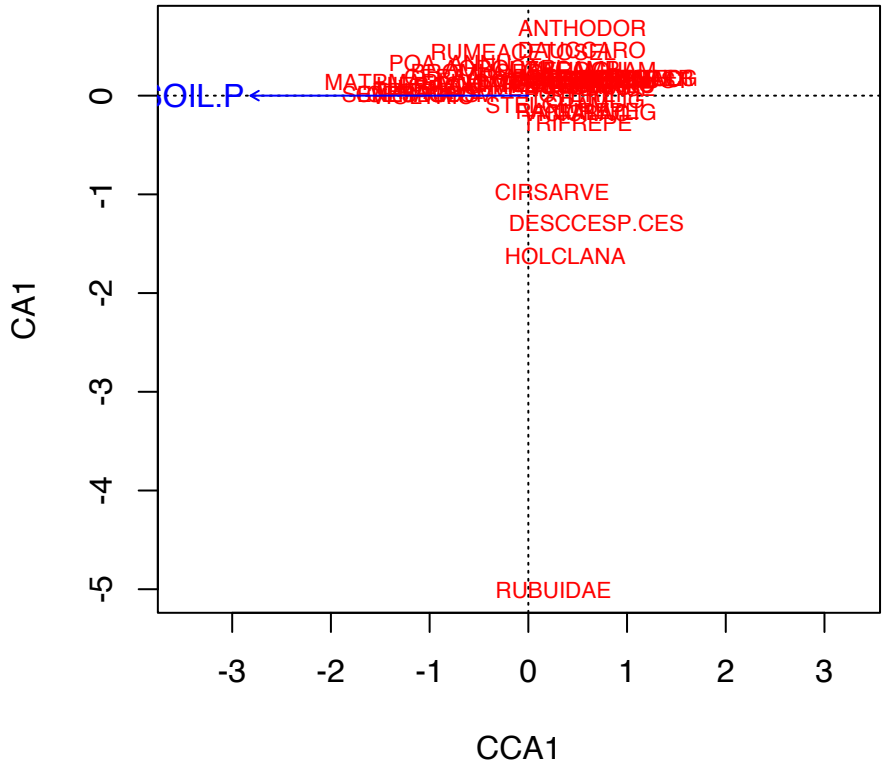
One of the main results from this CCA in which we might be interested, is the position for each species on the axes of the CCA ordination space. These are often referred to as *scores*, and hence the function provided in `vegan` also comes by the name `scores`. It returns a matrix where rows are species, and columns are the CCA axes. By default, `scores` returns the values for the first two axes only, no matter how many are actually calculated in the CCA.

```
head(vegan::scores(cca1, display = "species"))
```

```
##                CCA1       CCA2
## ACHIMILL  0.2380443 -0.3417982
## AGROCAPI  0.1546028 -0.2749053
## ANTHODOR  0.7269543  0.9151109
## BRIZMEDI -0.2597507  0.5466111
## BROMHORD -1.0215864 -1.3437564
## CAPSBURS  1.3432100 -0.5883097
```

Notice though that the position of the species on the axes does not directly correspond to the position on the actual gradients (which are represented in our plot as the arrows) but on the biplot shown above. Using the CCA axis scores therefore gives us only an indirect way of attributing species positions on the gradients, by relating the gradients to the CCA axis. For instance, we can determine the position of BROMHORD (the grass species ***Bromus hordeaceus***, soft brome) having a value of -1.02 on the first CCA1 axis, and conclude that it is a species that prefers grazed locations, but we cannot put actual values on the position along this environmental gradient. To do just that, de Bello et al. (2005) suggested to run separate CCAs (in the specific case in the paper it were RDAs) for each environmental variable, with other important environmental variables considered as (conditional) covariables, and then use scores on the first CCA axis to obtain the value for the particular gradient. This works because when a single environmental variable is fit in the CCA, the arrow of that variable is on the same dimension as the first CCA axis. We can see that when plotting the CCA with a single constraining variable, e.g. soil phosphor. The position of the species on that particular gradient can then be obtained from the scores of the first CCA axis. To still account for the effects of the other variables in the initial model (i.e.`cca1`, containing all environmental variables) we need to add the remaining variables as so called conditional variables (covariates).

```
cca_p <- cca(com ~ SOIL.P + Condition(dist.int + SOIL.WHC), data = env)
cca_m <- cca(com ~ SOIL.WHC + Condition(dist.int + SOIL.P), data = env)
cca_g <- cca(com ~ dist.int + Condition(SOIL.P + SOIL.WHC), data = env)
plot(cca_p, display = c("species", "bp"))
```




```
head(vegan::scores(cca_p, choice = 1, display = "species"))
```

```
##                CCA1
## ACHIMILL -0.4269343
## AGROCAPI -0.2606610
## ANTHODOR  0.5459867
## BRIZMEDI  1.0878676
## BROMHORD -0.5152544
## CAPSBURS -1.1228994
```

Once we obtained the species scores, we can investigate their relationship with the species traits, to understand if the traits we selected can help us to explain the position of the species along the studied environmental gradients. We come back to this point in section 4.6 of this material (RDA and regression trees). In the next section, we will first explore a method in which the traits are directly involved in the ordination.

4.4 Double Canonical Correspondance Analysis

Because it is a direct extension of the "trait-free" CCA, we will use so-called "double CCA" (dCCA) to demonstrate one possible way of analysis of species responses to the environment based on their traits. The first step in calculating dCCA is a correspondence analyses (CA), i.e. ordination of the community matrix, without involving the environmental variables as constrains (as we did by CCA). Rather than `cca` from vegan, we will use the `dudi.coa` function from the `ade4` package.

```
ca1 <- dudi.coa(com, scannf = F)
```

The result of this CA is then used in the `dbrda` function provided by Kleyer et al. (2012), to obtain the ordination of species that is constrained simultaneously by the species' traits and the environmental variables. Unfortunately, as is often the case in concepts and methodology,

the same abbreviations get used for different things. In this case, do not confuse this function `dbrda` with the function of the same name from the `vegan` package, which provides distance-based redundancy analyses. In the function we use here, *db* stands for double.

```
dCCA1 <- dbrda(ca1, env, traits, scannf = FALSE)
```

```
## Warning in scalewt(model.matrix(fmla, data = df1), weights1): Variables with
## null variance not standardized.
```

```
## Warning in scalewt(model.matrix(fmla, data = df2), weights2): Variables with
## null variance not standardized.
```

Package `ade4` does not provide similar summarizing and plotting functions as vegan, so we have to do things by hand. The percentage of the explained variation by both traits and environment is calculated as the ratio between summed eigenvalues of the dCCA and the CA (where the latter reflects the null hypothesis that no constrains are influencing the ordination of species).

```
sum(dCCA1$eig) / sum(ca1$eig)
```

```
## [1] 0.06248465
```

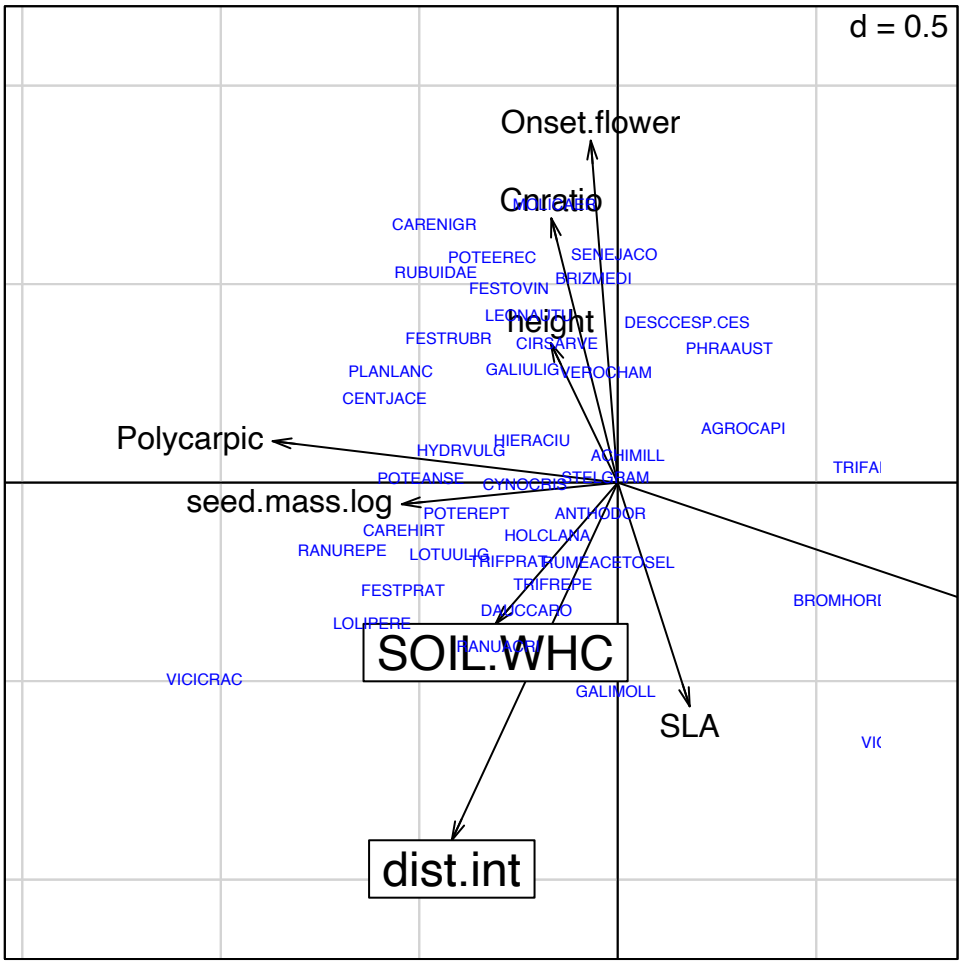
Decomposing the overall explained variation of 6.25% among the three axis we see that the two first axes explain the majority of this variation.

```
dCCA1$eig / sum(dCCA1$eig)
```

```
## [1] 0.58196117 0.38337723 0.03466159
```

As for the CCA, we can visualize our results in an ordination plot.

```
s.arrow(dCCA1$corZ[-1, ], ylim = c(-1.2, 1.2), boxes = FALSE)
s.arrow(dCCA1$corX[-1, ], add.plot = T, clab = 1.5)
pointLabel(dCCA1$co, row.names(dCCA1$co), cex = 0.5, col = "blue")
```



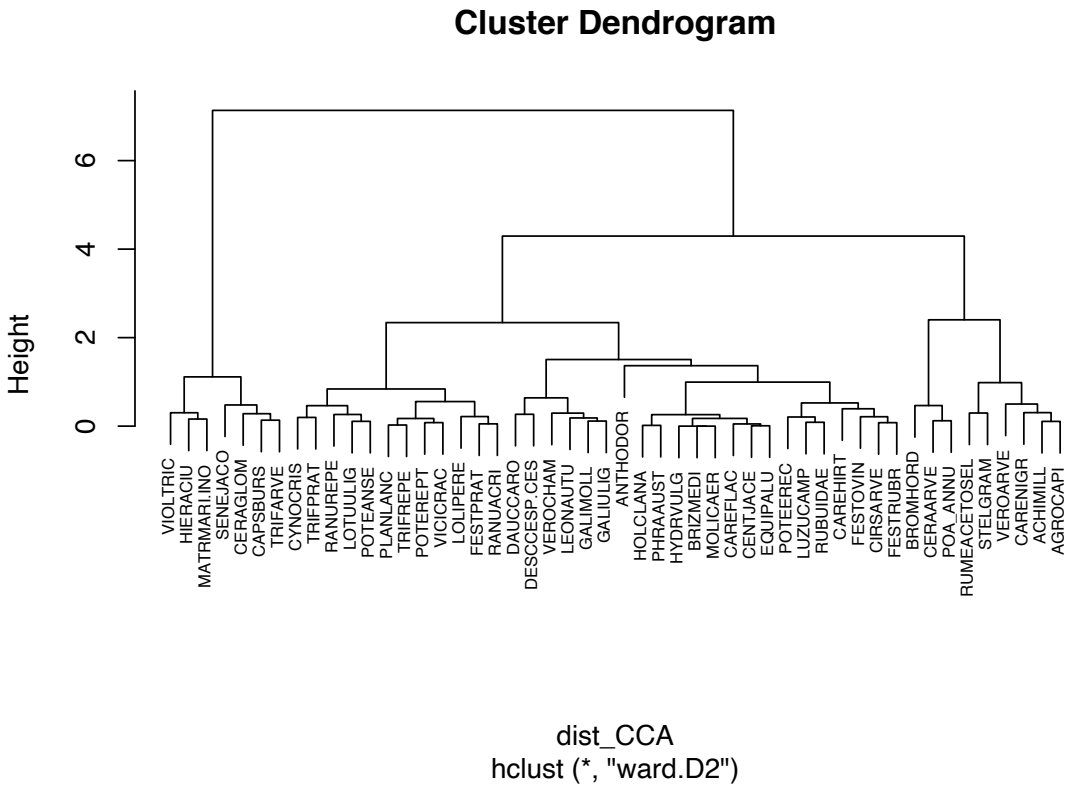
The interpretation of this ordination plot is very similar to the one we considered for the "trait-free" CA, but now it also involves the species' traits as an additional constraining factor. Polycarpic species and species with heavy seeds are negatively related with the first axis, translating into sites that have a low amount in available phosphorous and a high grazing intensity. On the second axis, high scores are representing tall species with low SLA values, high CN ratio and late onset of flowering, and these species prefer low grazing pressures. Through the approaches described so far, we can locate any of the species in the ordination space and therefore describe their position along the environmental gradients. In the next section we will make use of additional analyses to reveal groups of species with similar responses.

4.5 Functional response groups

After computing the type of ordinations shown above, another possible step in analyzing trait-environment relationships is the clustering of species according to their trait responses, i.e. creating *functional response groups* (see Chapter 3 in the reference book, Figure 3.5). Such an approach can aid, for instance, prioritization schemes in management planning, where it can be more feasible to group species according to responses to certain environmental drivers.

The creation of response groups, i.e. groups of species with similar environmental preferences (and traits), can be achieved in different ways. In the case of CCA (see section 4.4 above), we can use the scores already described and object them to clustering techniques.

```
dist_CCA <- dist(vegan::scores(cca1, display = "species"))
clust_CCA <- hclust(dist_CCA, method = "ward.D2")
plot(clust_CCA, cex = 0.6)
```



The graphical output we just plotted is a *cluster dendrogram*, i.e. a visual representation of the clustering algorithm we applied to the species scores, after these have been transformed into a distance matrix. Tough these dendrograms might look similar, do not confuse this kind of graphical output with the ones produced by regression trees, treated in the next section.

As described in R material Ch 3, we have the option to use statistical methods to help us decide on a reasonable number of clusters. The function `NbClust` from the package of the same name estimates the *optimal* number of groups on the basis of maximizing the dissimilarity between groups and minimizing the dissimilarity within groups.

```
groups_CCA <- NbClust(diss = dist_CCA, distance = NULL, min.nc = 2, max.nc = 6,
                      method = "ward.D2", index = "silhouette")
```

```
##
##  Only frey, mcclain, cindex, sihouette and dunn can be computed. To compute the other ind
```

```
groups_CCA
```

```
## $All.index
##      2      3      4      5      6
## 0.6280 0.5397 0.5055 0.4396 0.4421
##
## $Best.nc
## Number_clusters     Value_Index
##           2.000           0.628
```

```
##
## $Best.partition
##     ACHIMILL     AGROCAPI     ANTHODOR     BRIZMEDI     BROMHORD     CAPSBURS
##            1            1            1            1            1            2
##     CAREFLAC     CAREHIRT     CARENIGR     CENTJACE     CERAARVE     CERAGLOM
##            1            1            1            1            1            2
##     CIRSARVE     CYNOCRIS     DAUCCARO DESCCESP.CES     EQUIPALU     FESTOVIN
##            1            1            1            1            1            1
##     FESTPRAT     FESTRUBR     GALIMOLL     GALIULIG     HIERACIU     HOLCLANA
##            1            1            1            1            2            1
##     HYDRVULG     LEONAUTU     LOLIPERE     LOTUULIG     LUZUCAMP MATRMARI.INO
##            1            1            1            1            1            2
##     MOLICAER     PHRAAUST     PLANLANC     POA_ANNU     POTEANSE     POTEEREC
##            1            1            1            1            1            1
##     POTEREPT     RANUACRI     RANUREPE     RUBUIDAE RUMEACETOSEL     SENEJACO
##            1            1            1            1            1            2
##     STELGRAM     TRIFARVE     TRIFPRAT     TRIFREPE     VEROARVE     VEROCHAM
##            1            2            1            1            1            1
##     VICICRAC     VIOLTRIC
##            1            2
```

With the combination of clustering algorithm and index used in the function it proposes that two clusters are a good solution. The affiliation of our species to these two cluster are stored in the `$Best.partition` element of the output. Now, we can use our trait data to test if any of the traits are related to the two emerging clusters. In other words, we are asking whether there are traits that explain to which cluster the species belong, keeping in mind that this clustering is based on our initial ordination that arranges species along the considered environmental gradients. For this test we use here a simple regression analysis. When we provide a matrix of our traits on the left hand side of the formula (instead of a single trait), we get the entire set of regression outputs that test every trait against the cluster affiliation.

```
summary(lm(as.matrix(traits) ~ groups_CCA$Best.partition))
```

```
## Response Polycarpic :
##
## Call:
## lm(formula = Polycarpic ~ groups_CCA$Best.partition)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.93023  0.06977  0.06977  0.06977  0.71429
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  1.5748     0.1440  10.936 1.25e-14 ***
## groups_CCA$Best.partition  -0.6445     0.1208  -5.334 2.56e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2965 on 48 degrees of freedom
## Multiple R-squared:  0.3721, Adjusted R-squared:  0.3591
## F-statistic: 28.45 on 1 and 48 DF,  p-value: 2.564e-06
##
##
## Response Cnratio :
##
## Call:
```

```
## lm(formula = Cnratio ~ groups_CCA$Best.partition)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.370  -4.271  -1.840   3.446  19.862
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  18.114      3.368   5.378  2.2e-06 ***
## groups_CCA$Best.partition     1.201      2.826   0.425    0.673
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.935 on 48 degrees of freedom
## Multiple R-squared:  0.003746,   Adjusted R-squared:  -0.01701
## F-statistic: 0.1805 on 1 and 48 DF,  p-value: 0.6729
##
##
## Response seed.mass.log :
##
## Call:
## lm(formula = seed.mass.log ~ groups_CCA$Best.partition)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.60498 -0.26848  0.01702  0.29752  1.48402
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.2609     0.2919  -0.894    0.376
## groups_CCA$Best.partition   -0.1341     0.2449  -0.548    0.587
##
## Residual standard error: 0.6009 on 48 degrees of freedom
## Multiple R-squared:  0.006208,   Adjusted R-squared:  -0.0145
## F-statistic: 0.2999 on 1 and 48 DF,  p-value: 0.5865
##
##
## Response SLA :
##
## Call:
## lm(formula = SLA ~ groups_CCA$Best.partition)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.8160  -3.9602  -0.1145   4.2565  20.9340
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 23.1292     3.3195   6.968 8.24e-09 ***
## groups_CCA$Best.partition    0.6868     2.7857   0.247    0.806
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.835 on 48 degrees of freedom
## Multiple R-squared:  0.001265,   Adjusted R-squared:  -0.01954
## F-statistic: 0.06079 on 1 and 48 DF,  p-value: 0.8063
##
##
```

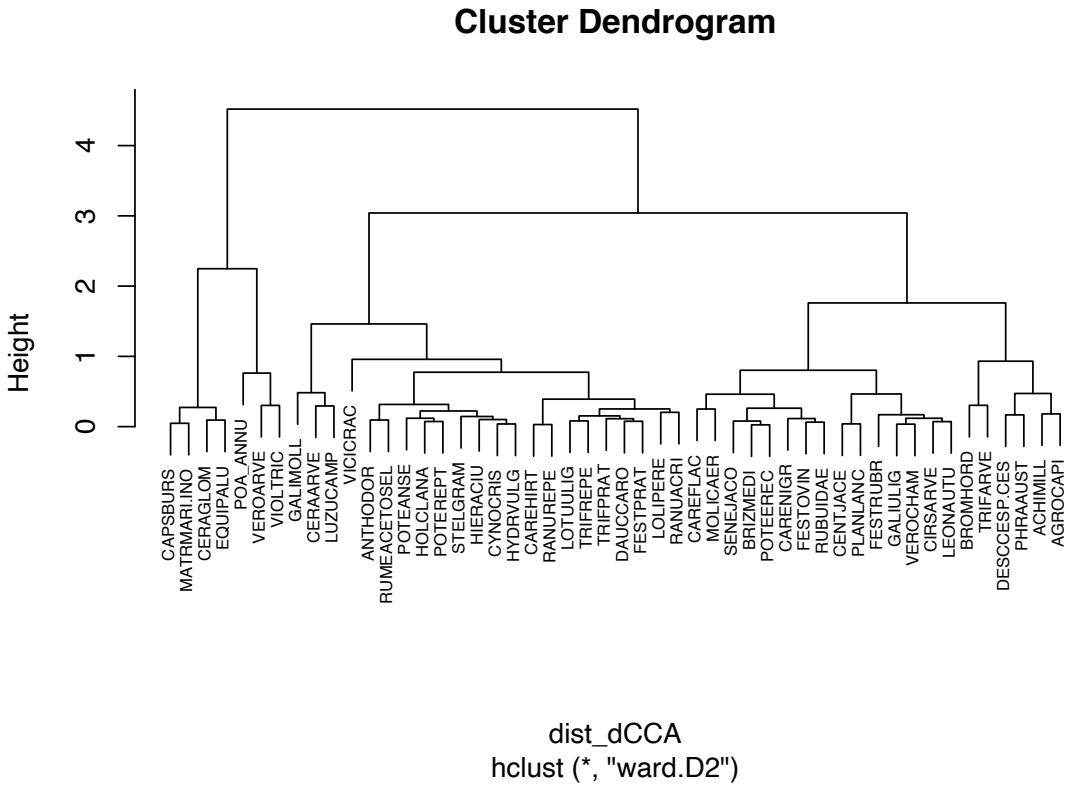
```
## Response height :
##
## Call:
## lm(formula = height ~ groups_CCA$Best.partition)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.466  -8.374  -3.307   3.357  42.234
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  24.429      6.698   3.647 0.000652 ***
## groups_CCA$Best.partition   -4.563      5.621  -0.812 0.420890
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.79 on 48 degrees of freedom
## Multiple R-squared:  0.01354,    Adjusted R-squared:  -0.007006
## F-statistic: 0.6591 on 1 and 48 DF,  p-value: 0.4209
##
##
## Response Onset.flower :
##
## Call:
## lm(formula = Onset.flower ~ groups_CCA$Best.partition)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -65.302  -2.052   4.698  12.448  31.698
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 161.890     11.747  13.781   <2e-16 ***
## groups_CCA$Best.partition    3.412      9.858   0.346    0.731
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.19 on 48 degrees of freedom
## Multiple R-squared:  0.00249,    Adjusted R-squared:  -0.01829
## F-statistic: 0.1198 on 1 and 48 DF,  p-value: 0.7308
```

We see that the trait `Polycarpic` is significantly associated to the affiliation of species in the two clusters. The majority of species in one of the clusters are monocarpic, i.e. they are annual species that die at the end of the vegetation period and only survive through their seeds. In the other cluster we mainly find species that are polycarpic, i.e. they live at least for several vegetation periods and flower and reproduce each year.

Notice that the example shown here is somewhat similar to the one shown for creating functional groups of students based on their *traits* in R material Ch 3. However in that case we first considered the traits and made the clusters based on these traits. Theoretically, we could then test if the "species" (i.e. individual students) in these functional trait groups have different "environmental preferences". Hence, functional groups are created first, and then their response to environmental gradients is analyzed (as in cluster regression, see Kleyer et al., 2012). Above, on the contrary, we first made groups based on environmental preferences (through the CCA ordination) and then related these to the traits. Sometimes the first approach is called 'a priori functional groups', because first the groups are created based on an 'a priori' selection of traits. The second approach is called 'a posteriori', as the relation between species preferences with traits is analyzed after the creation of response groups.

The approach of using dCCA presents a special case, since both traits and environmental gradients are simultaneously involved in the ordination. From dCCA, once we obtained the position of species in the ordination space constrained by their traits and the environment, we can use these positions (i.e. the ordination scores) in a standard cluster analyses to obtain the functional response groups.

```
dist_dCCA <- dist(dCCA1$co)
clust_dCCA <- hclust(dist_dCCA, method = "ward.D2")
plot(clust_dCCA, cex = 0.6)
```



Following the same methodology as above, we use `NbClust` to determine the number of clusters in our ordination.

```
groups_dCCA <- NbClust(diss = dist_dCCA, distance = NULL, min.nc = 2, max.nc = 9,
                       method = "ward.D2", index = "silhouette")
```

```
##
##  Only frey, mcclain, cindex, sihouette and dunn can be computed. To compute the other ind
```

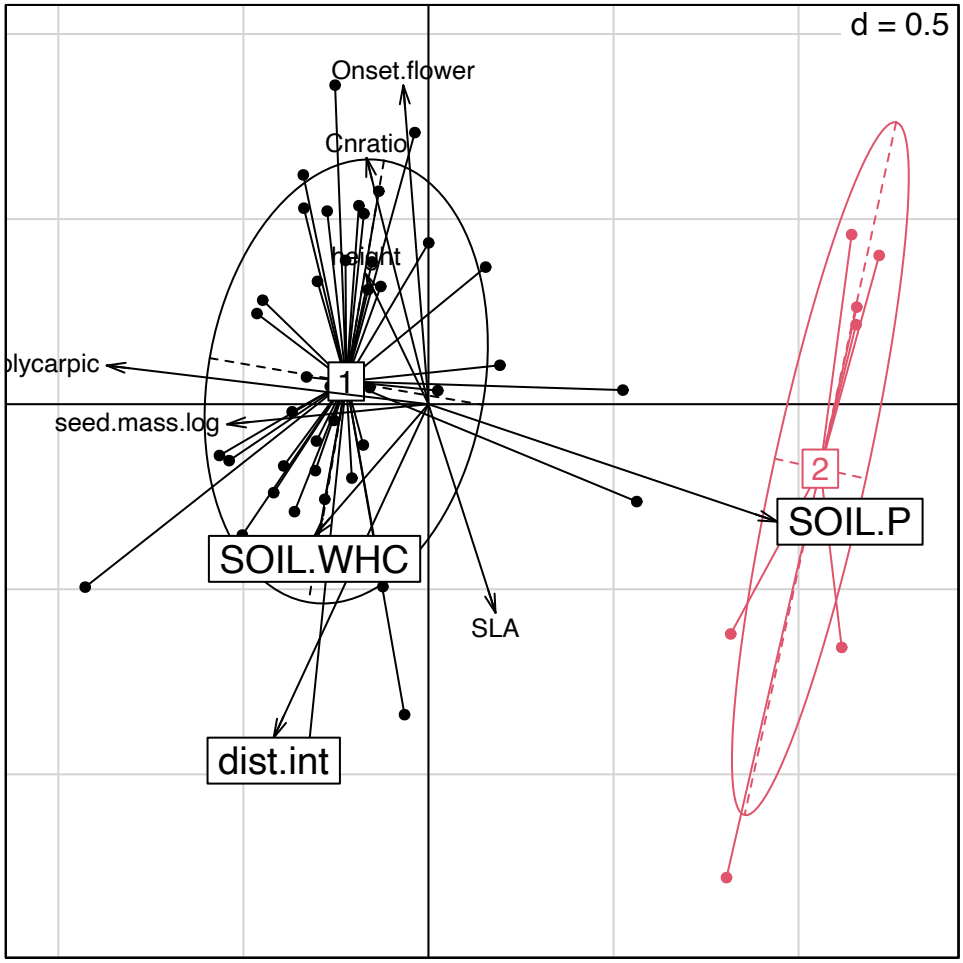
```
groups_dCCA
```

```
## $All.index
##      2      3      4      5      6      7      8      9
## 0.5721 0.4315 0.4684 0.4776 0.4830 0.5016 0.4641 0.4406
##
## $Best.nc
## Number_clusters     Value_Index
##          2.0000          0.5721
##
## $Best.partition
```

```
##     ACHIMILL     AGROCAPI     ANTHODOR     BRIZMEDI     BROMHORD     CAPSBURS
##            1            1            1            1            1            2
##     CAREFLAC     CAREHIRT     CARENIGR     CENTJACE     CERAARVE     CERAGLOM
##            1            1            1            1            1            2
##     CIRSARVE     CYNOCRIS     DAUCCARO DESCCESP.CES     EQUIPALU     FESTOVIN
##            1            1            1            1            2            1
##     FESTPRAT     FESTRUBR     GALIMOLL     GALIULIG     HIERACIU     HOLCLANA
##            1            1            1            1            1            1
##     HYDRVULG     LEONAUTU     LOLIPERE     LOTUULIG     LUZUCAMP MATRMARI.INO
##            1            1            1            1            1            2
##     MOLICAER     PHRAAUST     PLANLANC     POA_ANNU     POTEANSE     POTEEREC
##            1            1            1            2            1            1
##     POTEREPT     RANUACRI     RANUREPE     RUBUIDAE RUMEACETOSEL     SENEJACO
##            1            1            1            1            1            1
##     STELGRAM     TRIFARVE     TRIFPRAT     TRIFREPE     VEROARVE     VEROCHAM
##            1            1            1            1            2            1
##     VICICRAC     VIOLTRIC
##            1            2
```

Remember that in the dCCA approach the species are already ordered in the ordination space considering both, the environmental variable and the species traits. We can also visualize the ordination, taking into account the two clusters the species form.

```
clusters <- as.factor(groups_dCCA$Best.partition)
s.class(dCCA1$co, clusters, col = 1:nlevels(clusters))
s.arrow(dCCA1$corZ[-1,], add.plot = T, clab = 0.8, boxes = FALSE)
s.arrow(dCCA1$corX[-1, ], add.plot = T, clab = 1.2)
```



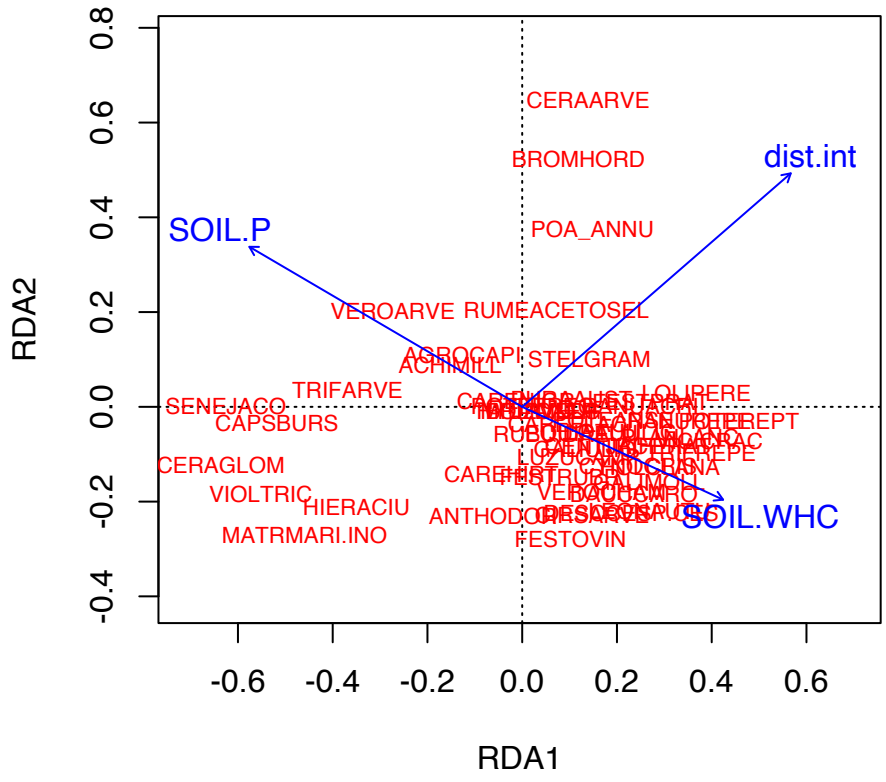
Not surprisingly, since based on the same data, we get a similar result from the one using CCA, i.e. relating the traits to the gradients *after* the ordination and subsequent clustering. In the plot here we can see that the two groups separate along the first axis, which at the same time is the axis that is correlated with the `Polycarpic` trait. As before, species in cluster 2 are mostly annuals, and species in cluster 1 mostly perennials.

4.6 RDA and regression trees

To present another method that statistically works differently from dCCA, we focus on a combination of RDA and regression trees. Keep in mind though that the purpose of these different methods is always the same - to model species responses to environmental gradients through their traits.

The first step here uses RDA to ordinate the species constrained by the environment, i.e. we obtain a position of each species along the RDA axes, and thus along the environmental gradients.

```
rda1 <- rda(com ~ ., data = env, scale = TRUE)
plot(rda1, display = c("bp", "sp"))
```



Comparing the RDA plot with that from the CCA output, we can see that they are quite similar. This is not surprising, since both methods try to achieve the same thing, i.e. maximize the explained variability in species responses through environmental variables.

> Be aware though that RDA is generally considered to be more appropriate when environmental gradients are 'short' and the relationship between the species abundances and the gradients is linear. CCA on the other hand is the more adequate choice when environmental gradients are 'large', and we expect unimodal relationships of the species with the gradients, i.e. the species appear and disappear across the entire length of the gradients, having an optimum somewhere along. For the sake of demonstration, we neglect here this distinction between

RDA and CCA, and introduce both methods with the same data set, not making any assumption about how species are related to the gradients (linear or unimodal).

Let us return to the actual analyses at hand, and look at the adjusted R^2 of the RDA we just computed. We can see that even the explained variabilities are quite similar across the two methods of RDA and CCA.

```
RsquareAdj(rda1)
```

```
## $r.squared
## [1] 0.145125
##
## $adj.r.squared
## [1] 0.07936543
```

The predictive power of the model is similar to the CCA. We can then look how much of the variation the 3 constraining factors explain.

```
cumsum(rda1$CCA$eig) / sum(rda1$CCA$eig)
```

```
##      RDA1      RDA2      RDA3
## 0.5685573 0.8130239 1.0000000
```

Because 80 % of this explained variability (i.e. the 8%) can be attributed to the first two RDA axes, we only take into account the first two axis for the next step, running a regression tree.

The traits of the species are then used to model each species response (species' scores), in this case defined by the RDA. This can be done, for instance with a linear regression or, as was suggested, by regression trees. We will first show a multiple linear regression of the species responses with the traits. Then we will use the opportunity to introduce regression trees, starting with a relatively simple model. In the next section we will delve a bit deeper into the topic of regression trees, particularly showing boosted regression tree.

Analyzing the relationship between the species responses (expressed as the species scores in the RDA ordination space) and their traits is relatively straight forward. For this purpose, we use the species scores from the RDA as a response, and the traits of the species as predictors in a multiple linear model (`lm`, of which we then return the `summary`).

```
lm_rda <- lm(vegan::scores(rda1, choices = 1, display = "species") ~ .,  data = traits)
summary(lm_rda)
```

```
##
## Call:
## lm(formula = vegan::scores(rda1, choices = 1, display = "species") ~
##     ., data = traits)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.64119 -0.09287  0.02088  0.11510  0.34581
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.225e-01  3.271e-01   0.680   0.5000
## Polycarpic     4.909e-01  8.184e-02   5.999 3.67e-07 ***
## Cnratio       -1.148e-02  4.426e-03  -2.594   0.0129 *
```

```
## seed.mass.log  1.164e-01  4.777e-02   2.437   0.0190 *
## SLA           -1.451e-03  5.403e-03  -0.269   0.7896
## height         3.255e-05  2.442e-03   0.013   0.9894
## Onset.flower  -1.635e-03  1.410e-03  -1.159   0.2527
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1974 on 43 degrees of freedom
## Multiple R-squared:  0.5363, Adjusted R-squared:  0.4716
## F-statistic: 8.289 on 6 and 43 DF,  p-value: 5.474e-06
```

In the output we can see that the traits of our species explain a significant proportion of the species responses, 47%. Species being either polycarpic or monocarpic is an important determinant, with the C/N ratio and seed mass also identified as significant factors. Note two important differences with the analyses of response groups conducted above. First, as we did not cluster species into groups, we are now modeling the response of each species, not of groups of species. Second, while in the response groups example our response were the traits, now the traits are the explanatory variable. Despite these differences, both methods identified the Polycarpic trait as important to explain the species position along the environmental gradients.

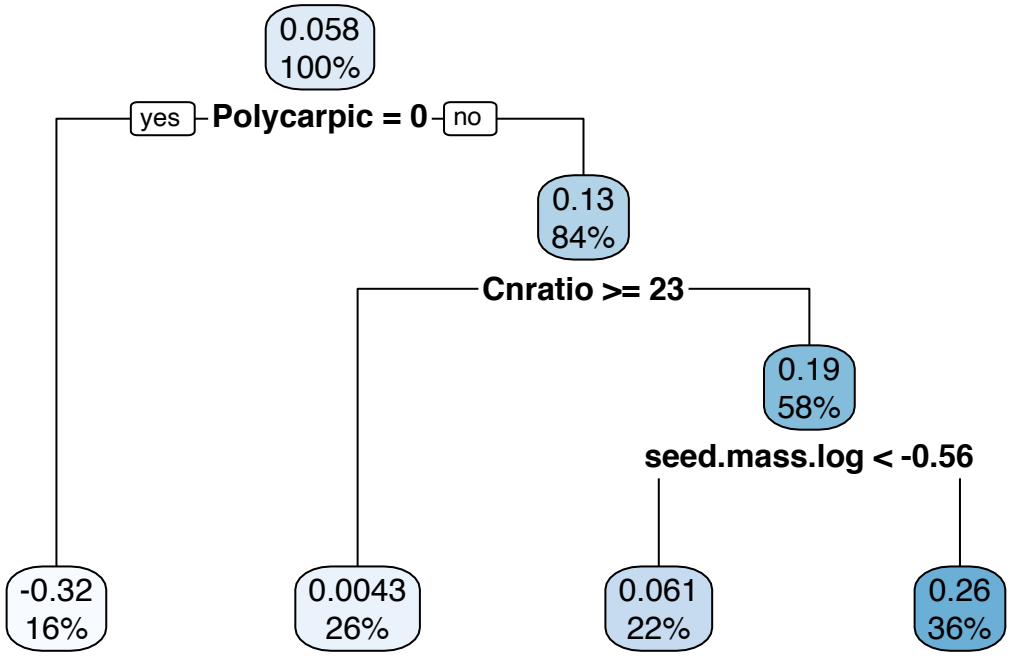
Let us now turn our attention to regression trees. Regression trees are a different statistical method compared to linear regression, even though they are still based on regressing the data. Regression trees are built by recursive partitioning, which tries to hierarchically split the data into "branches". At each branch, the response is split by one of the predictors. E.g., at the topmost branch, a particular trait value splits the response variable into two groups, according to if the trait value of a species is higher or lower than that value. Both of these new groups are then potentially split again by some other or the same trait. Predictors can be continuous and categorical, and once a predictor has defined a branch in the tree, it can occur again at lower branches. This allows for non-additive effects of traits, which in normal linear models can only be added by interactions (which can quickly become tedious and unpractical, because of the large number of possible interactions when many explanatory variables are involved). Once the recursive partitioning does not find any new rules that can split the branches further in a meaningful way, the resulting tree is obtained. This sounds a bit abstract, so let's look at a concrete example with our data. For this, first we create a new data frame that contains our response, i.e. the scores of the species in the RDA ordination space of the RDA we ran previously.

```r
# Create a data frame that contains both, the response (i.e. species scores from
# RDA axes 1 and 2) and the traits to model these responses.
scores_rda1 <- vegan::scores(rda1, choices = c(1, 2))$species
df_response <- cbind(scores_rda1, traits)
head(df_response)
```

```
##                 RDA1         RDA2 Polycarpic Cnratio seed.mass.log   SLA height
## ACHIMILL -0.15211025  0.088148471          1  12.940        -0.876 19.63 21.150
## AGROCAPI -0.12636683  0.108928310          1  17.421        -1.444 29.54 21.017
## ANTHODOR -0.06157151 -0.230064992          1  13.778        -0.678 29.97 14.967
## BRIZMEDI  0.03526923 -0.009676056          1  33.690        -0.578 25.93 20.167
## BROMHORD  0.11888772  0.522135516          0  16.183         0.305 26.19 12.367
## CAPSBURS -0.51768745 -0.033843271          0  11.624        -0.996 24.94 15.267
##          Onset.flower
## ACHIMILL          178
## AGROCAPI          167
## ANTHODOR          170
## BRIZMEDI          163
## BROMHORD          160
## CAPSBURS          198
```

The trait data is now accompanied by the scores on the first two axes of the RDA, labeled `RDA1` and `RDA2`. We can now construct our first regression tree. As you can see in the following code, we can specify our regression tree model in the usual formula style, i.e. as if we would set up a linear regression model with 'lm'. We can then plot the result, which is a representation of the "rules" that split the data. The following model is done using the species scores on the first RDA axis as response.

```
# Run a multivariate regression tree with function rpart.
rta <- rpart::rpart(RDA1 ~ Polycarpic + Cnratio + seed.mass.log + SLA +
                      height + Onset.flower, data = df_response)
rpart.plot(rta)
```



As mentioned above, the plotted figure represents a series of partition rules, and we read it from the top of the tree to the bottom. For our response (i.e. the position of species along the first RDA axis), the trait `Polycarpic` is the most important explanatory variable; but what do the other values mean? The box tells us something about the response variable. Note that here we only used the scores from the first axis. We can read from the box that the predicted overall mean of these scores (i.e. of 100% of the species included in the model) is 0.058. `Polycarpic` is a binary trait (i.e. a species is either polycarpic or not), and we can see that when the value is 0 (i.e. the species is monocarpic, rather than polycarpic), the mean of the scores of these species is -0.32, and they represent 16% of all species. Remember that, according to our RDA, negative values represent species on the left hand side of the RDA ordination we plotted above, i.e. the appear in localities with relatively high soil phosphate, and little soil moisture and grazing. The remaining 84% of species, with a mean of 0.13 for the scores, can be split further, according to additional traits. The same logic applies to these splits, just that now our conditions are not binary, but define whether a species has a trait value smaller or bigger than the split. Following the top to bottom direction, polycarpic species are split into species whose leaf C/N ratio is either bigger, or smaller than (or equal to) 23. In the former case, the predicted mean is 0.0043, and this group includes 26% of our species. The species that have a leaf C/N smaller or equal to 23 are further split. This time, seed mass leads to the last split in our tree, with species that have seeds lighter than -0.56 (on a logarithmic scale) predicted to have a mean axis score of 0.061 (22% of species), and species heavier than that with a predicted mean of 0.26 (36% of species).

It is obvious that this type of results we get from a regression tree is quite a bit different from the kind of results that we usually get from other types of regression models, in particular results from (non)linear regression models. There are no slope estimates and significance

values here for the single variables, no "model fit" (such as R^2) is reported, and we do not fit interactions in an explicit way. But we can interpret the outputs as having "interacting" traits in a more general way, i.e. the trait CN ratio is only important once the Polycarpic trait is taken into consideration, and seed mass is only important after the response has been split according to CN ratio. However, note that at the same time the traits identified as important by the regression tree, are the same ones that were identified via the linear model.

4.7 Boosted regression trees (BRT)

At the end of the previous section we have already met a simple method of what is generally known as regression trees. Regression trees exist in different varieties of different complexity, and they offer some advantages to other regression-based models. One recently introduced way to make use of regression trees is in fact the attempt to use functional traits to model the fitness of species, i.e. returning to and testing one of the very premises of trait-based ecology (see Chapter 2). To familiarize you with the approach of so-called boosted regression trees, we will use an abbreviated version of the appendix to Pistón et al. (2019) The original appendix is going more into some of the details, and we encourage you to explore it in case you want to dive deeper into this method. We particularly leave out here the actual algorithm behind creating the splitting rules and instead highlight the advantages (but also disadvantages) of regression trees, and give an introduction to how we can use machine learning techniques to aggregate the results of more than one regression tree model based on the same data.

The data we are using (and already loaded in the data section at the beginning of this R material) is a subset of the data used in the Pistón et al. (2019) paper. For the sake of demonstration, we are only going to use a selection of traits contained there. In particular, we are going to use a variable called `veget` to describe the fitness of the species in terms of vegetative expansion, and the traits `SLA` (specific leaf area) and `SM` (seed mass) to model this fitness.

The data we are using (and already loaded in the data section at the beginning of this R material) is a subset of the data used in the Pistón et al. (2019) paper. For the sake of demonstration, we are only going to use a selection of traits contained there. In particular, we are going to use a variable called `veget` to describe the fitness of the species in terms of vegetative expansion, and the traits `SLA` (specific leaf area) and `SM` (seed mass) to model this fitness.

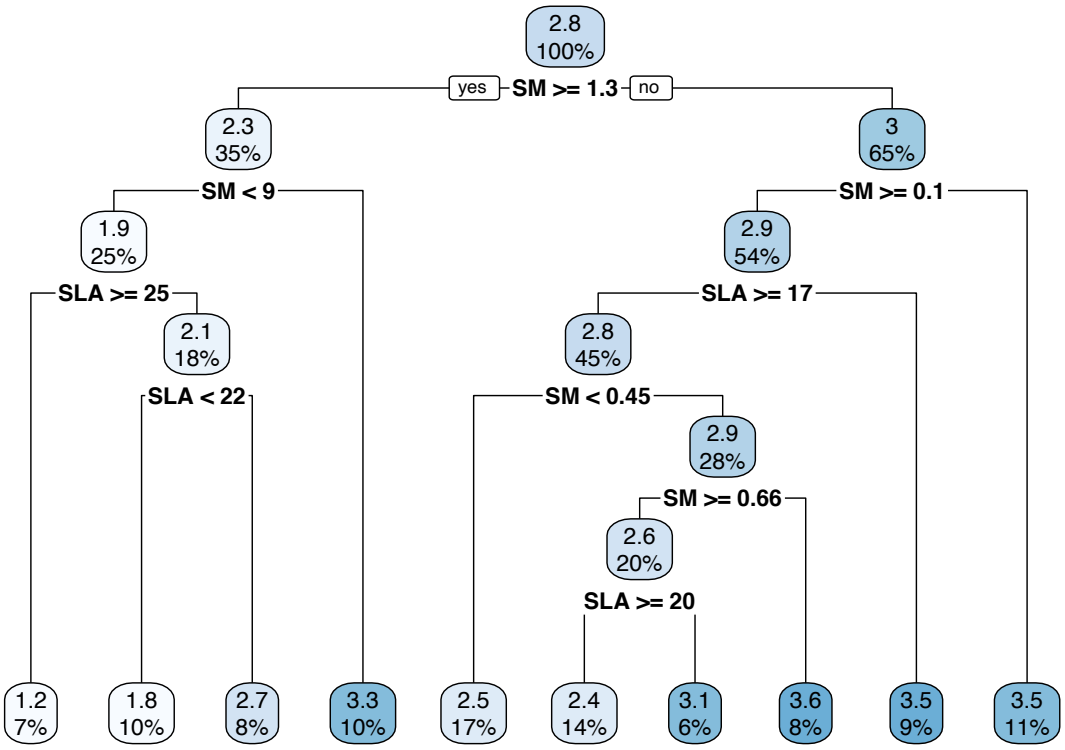
```
head(meadows)
```

```
##                      species veget        PH       TLA          SM      SLA SLS
## 1 Achillea_millefolium_agg.     3 0.3977500  411.0000 0.07000000 18.93521   2
## 2          Achillea_nobilis     4 0.4000000        NA 0.06800000       NA   2
## 3         Achillea_ptarmica     4 0.7775000  119.0000 0.15325000 16.29581   1
## 4         Aconitum_plicatum     1 0.8125000 5535.6250 4.42500000 21.11000   1
## 5       Agrimonia_eupatoria     3 0.3316667 8461.0000 4.23000000 18.45000   1
## 6      Agrostis_stolonifera     4 0.4500000  365.6667 0.03953333 31.41949   2
##   LH    P        LS   CGO     BB.D     BB.S
## 1 pc 4.00 0.1425694 4.750 5.741935 22.14286
## 2 pc 3.50 0.0987500 3.375 8.920354 37.66667
## 3 pc 4.00 0.1300000 6.000 4.750000 20.00000
## 4 pc 1.00 0.0050000 1.000 3.500000 10.00000
## 5 pc 4.00 0.0300000 1.000 4.750000 20.00000
## 6 pc 1.75 0.1300000 5.000 2.000000  6.25000
```

But first of all, we will turn to the important question of how does the regression tree model actually know when to stop splitting the data?

Simply speaking, under the default setting, `rpart` keeps partitioning the space made up by the explanatory variables until there are too few observations to make further splits. In the case of our data relating demographic parameters of species to their functional traits, we attain a very deep tree, like the one we show below:

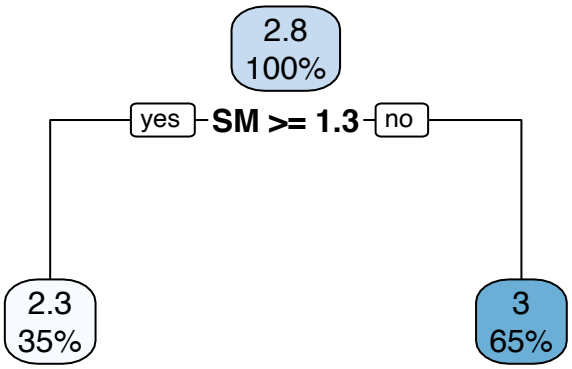
```
treeVegetFull <- rpart(veget ~ SLA + SM, data = meadows)
rpart.plot(treeVegetFull)
```



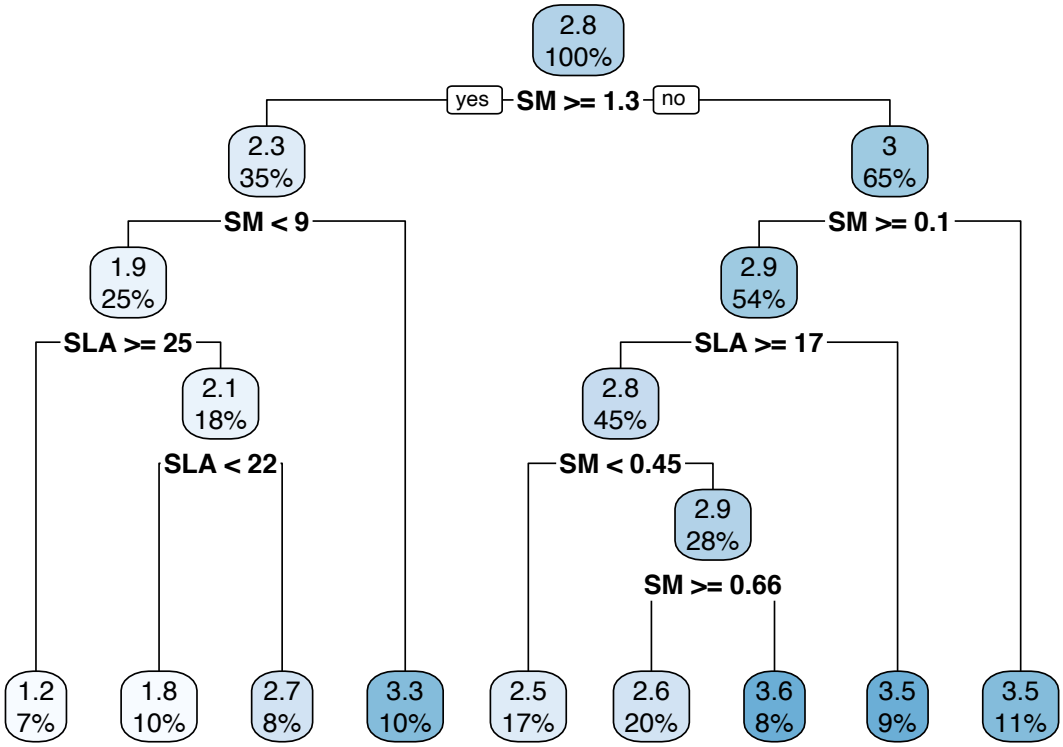
The resulting regression tree has a lot of splits that partition the species into many groups. That is an indication that we have *overfitted* our tree. What does this mean? Perhaps our tree is very well adjusted to the particularities in our dataset, but maybe it is not such a good predictor of `veget` for species (or individuals) that are not included in the dataset. As an extreme case of this, imagine a tree in which all its tips are occupied by a single observation. In general, we want to avoid this to happen. Because of that, trees are usually *pruned* by means of cross validation processes, removing some of the least important *branches* (particularly those with too few observations). In the first tree that we adjusted, we did something similar with the parameter cp, which basically tells the tree not to incorporate a new partition unless the R^2 of the model increases at least cp units (in our case 0.05).

There is another control parameter that will be interesting to present now, because we will use it later, which is `maxdepth`. This argument sets the maximum depth of any node of the tree, with 1 meaning that there is only one node (one decision).

```
treeVeget1 <- rpart(veget ~ SLA + SM, data = meadows,
                    control = rpart.control(maxdepth = 1))
rpart.plot(treeVeget1)
```




```
treeVeget5 <- rpart(veget ~ SLA + SM, data = meadows,
                    control = rpart.control(maxdepth = 5))
rpart.plot(treeVeget5)
```



4.7.1 How useful are regression trees?

Regression trees have some advantages that makes them an interesting tool to consider in our analyses:

- They are rather easy to understand and interpret once familiar with them.
- They can be easily plotted, and the representation is easy, even for non-experts.
- Qualitative predictors are incorporated easily, with no need to create dummy variables.
- They automatically incorporate *interactions* between the predictors, and they do not assume any a priori relationship between the variables.
- They are robust to outliers in the predictors. And also to transformations.
- Observations with some missing values in the predictors are still used, unlike in traditional regression models. This is particularly important when working with trait databases (as we emphasize below and in the main text).

The main problem of regression trees is that they generally do not have a great predictive power, particularly when compared with other regression and classification methods. Nevertheless, methods that aggregate trees by means of resampling techniques (such as *random forests* and *boosted trees*) can substantially improve the predictive performance of these models – although this comes with the cost of a considerable reduction in the interpretability of the model. In the remaining part of this document we will explore these techniques, with a particular focus on the analyses performed in the paper. We can start by explaining what this aggregation is based on.

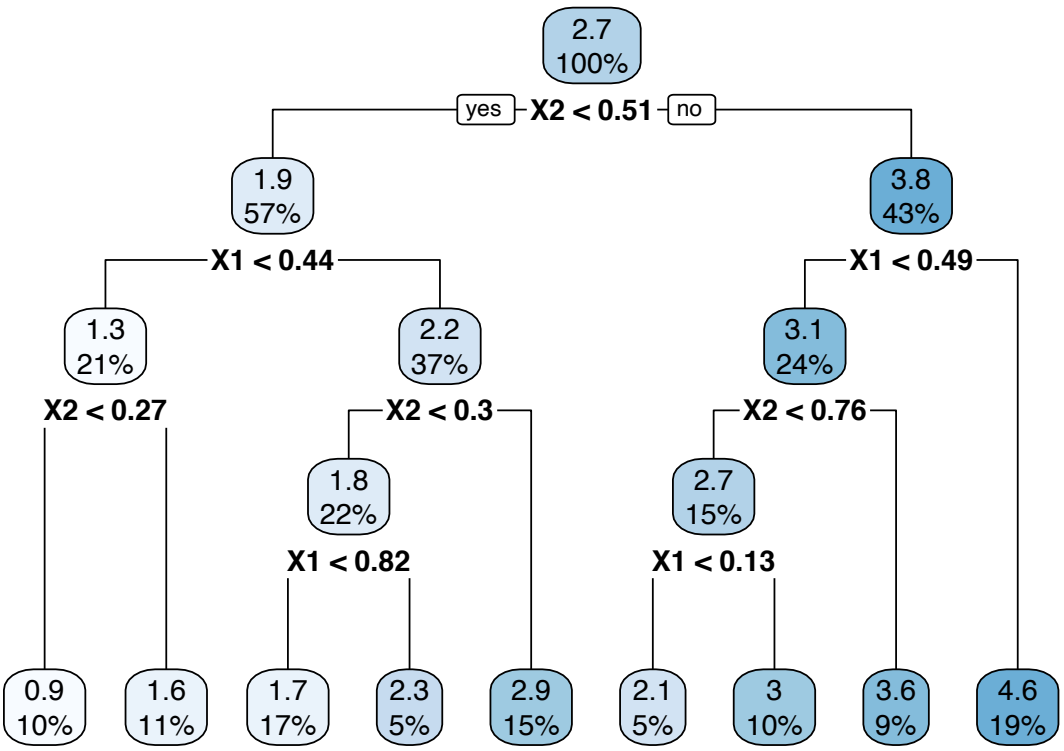
4.7.2 *Aggregating models*

Regression trees are very sensitive to changes in the data used to adjust them. This property in statistics is termed *variance*: regression trees have high variance. Let us examine this with invented data where our response variable (Y) will be a function of two interacting predictors (X1 and X2), so that $Y = 2(X1) + 3X2 + (X1 * X2) + \epsilon$:

```
set.seed(1)
size <- 150
X1 <- runif(size, 0, 1)
X2 <- runif(size, 0, 1)
eps <- rnorm(size, 0, 0.5)
Y <- 2 * X1 + 3 * X2 + (X1 * X2) + eps
data <- data.frame(Y = Y, X1 = X1, X2 = X2)
```

We can fit both a tree and a linear regression model to the dataset:

```
treeData <- rpart(Y ~ X1 + X2, data = data, control = rpart.control(cp = 0.01))
rpart.plot(treeData)
```




```
lmData <- lm(Y ~ X1 * X2, data = data)
summary(lmData)
```

```
##
## Call:
## lm(formula = Y ~ X1 * X2, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.34583 -0.31607 -0.01059  0.33674  1.28725
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.08854    0.18068   0.490  0.62485
## X1           1.67653    0.30285   5.536 1.40e-07 ***
## X2           2.93225    0.33302   8.805 3.47e-15 ***
## X1:X2        1.55746    0.56920   2.736  0.00699 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5106 on 146 degrees of freedom
## Multiple R-squared:  0.8497, Adjusted R-squared:  0.8467
## F-statistic: 275.2 on 3 and 146 DF,  p-value: < 2.2e-16
```

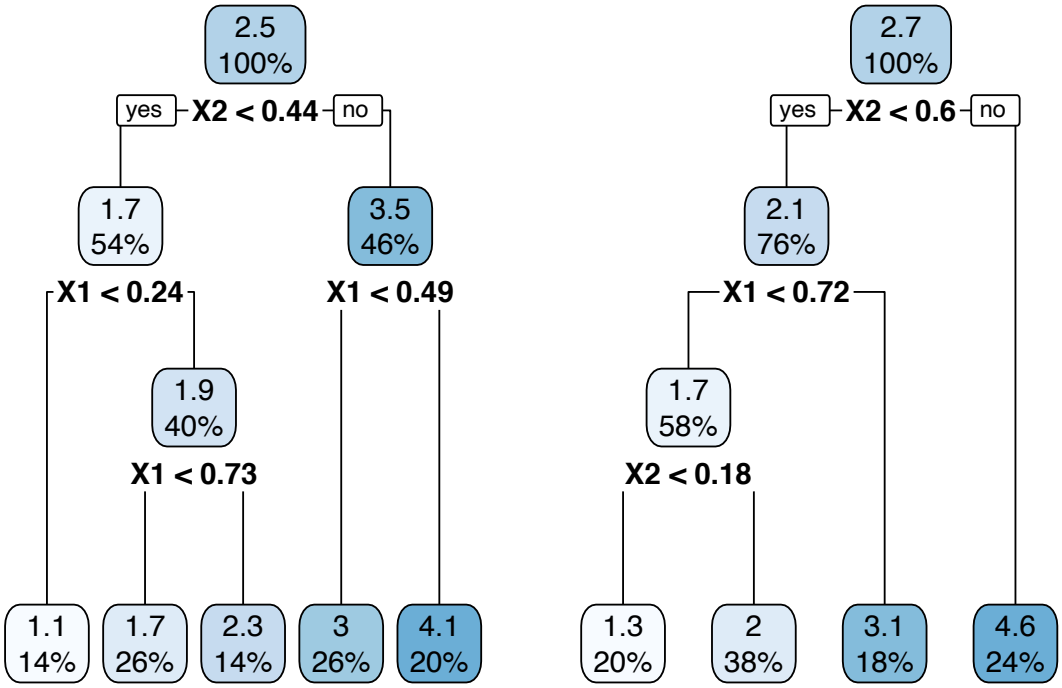
Note that for the linear model we have to specify the interaction term by using the asterisk * between the model terms, whereas with the regression tree the interactions are inherent in the model and do not need to be explicitly stated in the model formula. However, it is easy to see that the depth of the tree somehow allows interaction between predictors to play a role.

To illustrate the variance idea, we are going to divide our dataset into three parts (A, B, and T). We will fit regression trees using A and B and will use T to test these models. To do this, we will randomly extract one third of the rows to make T, and with the rest we will choose even rows for A and odd ones for B:

```
rowsT <- sample(1:nrow(data), floor(nrow(data) / 3))
dataT <- data[rowsT, ]
dataNoT <- data[-rowsT, ]
even <- seq(2, nrow(dataNoT), by = 2)
odd <- seq(1, nrow(dataNoT), by = 2)
dataA <- dataNoT[even,]
dataB <- dataNoT[odd,]
treeA <- rpart(Y ~ X1 + X2, data = dataA, control = rpart.control(cp = 0.001))
treeB <- rpart(Y ~ X1 + X2, data = dataB, control = rpart.control(cp = 0.001))
```

Do the models differ very much?

```
par(mfrow = c(1, 2))
rpart.plot(treeA)
rpart.plot(treeB)
```



We can see they are dissimilar! The partition rules in each tree are different, so that they differ in the values of the response variables at which the partitions are made. In more extreme cases, even the predictor in which each node is based changes between models, among other differences. What if we would have built two linear regressions?

```
lmA <- lm(Y ~ X1 * X2, data = dataA)
lmB <- lm(Y ~ X1 * X2, data = dataB)

summary(lmA)
```

```
##
## Call:
## lm(formula = Y ~ X1 * X2, data = dataA)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.28163 -0.27073  0.03162  0.27569  0.90935
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.2037     0.2451   0.831 0.410357
## X1            1.5958     0.4108   3.885 0.000326 ***
## X2            2.7586     0.4895   5.636 1.01e-06 ***
## X1:X2         1.5978     0.8633   1.851 0.070618 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4593 on 46 degrees of freedom
## Multiple R-squared:  0.8653, Adjusted R-squared:  0.8565
## F-statistic: 98.47 on 3 and 46 DF,  p-value: < 2.2e-16
```

```
summary(lmB)
```

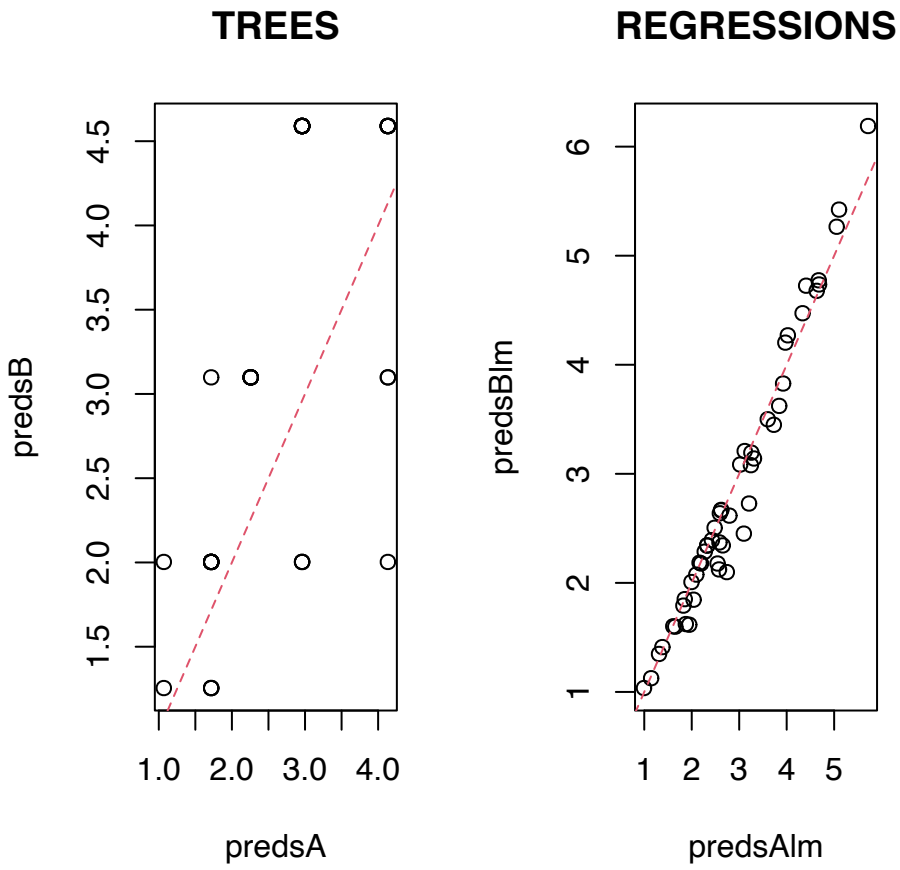
```
##
## Call:
```

```
## lm(formula = Y ~ X1 * X2, data = dataB)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.16784 -0.38404  0.05897  0.35798  1.09080
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4309     0.3215   1.340  0.18673
## X1            1.2286     0.5362   2.291  0.02657 *
## X2            1.7243     0.7513   2.295  0.02633 *
## X1:X2         3.3941     1.1971   2.835  0.00678 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.499 on 46 degrees of freedom
## Multiple R-squared:  0.8748, Adjusted R-squared:  0.8667
## F-statistic: 107.2 on 3 and 46 DF,  p-value: < 2.2e-16
```

Just by examining the coefficients it seems as if the linear models are much more consistent between them than the trees. This is because linear models tend to have lower variance than regression trees.

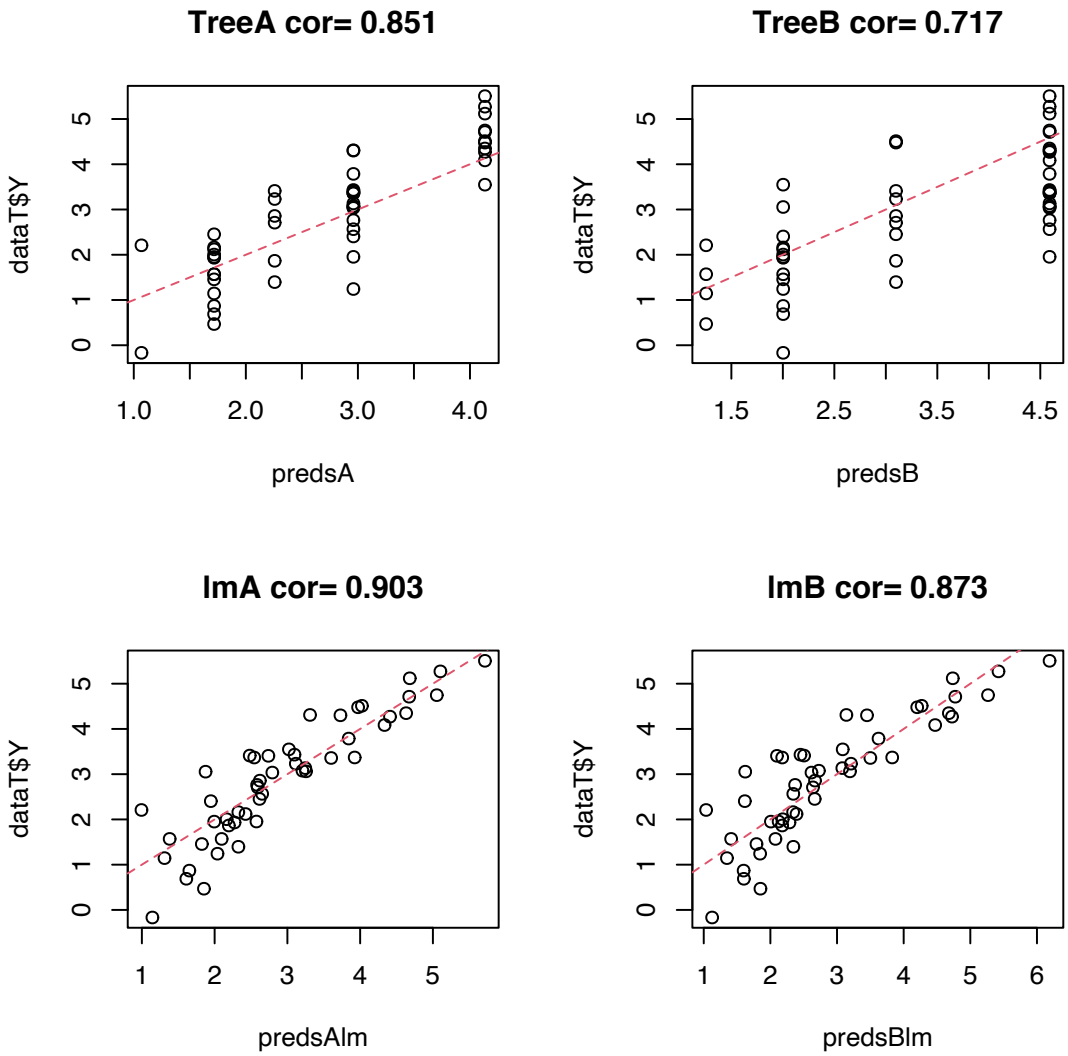
Since we have set one third of the data apart, we can see what do our models predicts for this testing subset.

```r
#First, the trees:
predsA <- predict(treeA, newdata = dataT)
predsB <- predict(treeB, newdata = dataT)

par(mfrow = c(1, 2))
plot(predsA, predsB, main = "TREES")
abline(0, 1, lty = 2, col = 2)

#Then regressions:

predsAlm <- predict(lmA, newdata = dataT)
predsBlm <- predict(lmB, newdata = dataT)

plot(predsAlm, predsBlm, main = "REGRESSIONS")
abline(0, 1, lty = 2, col = 2)
```



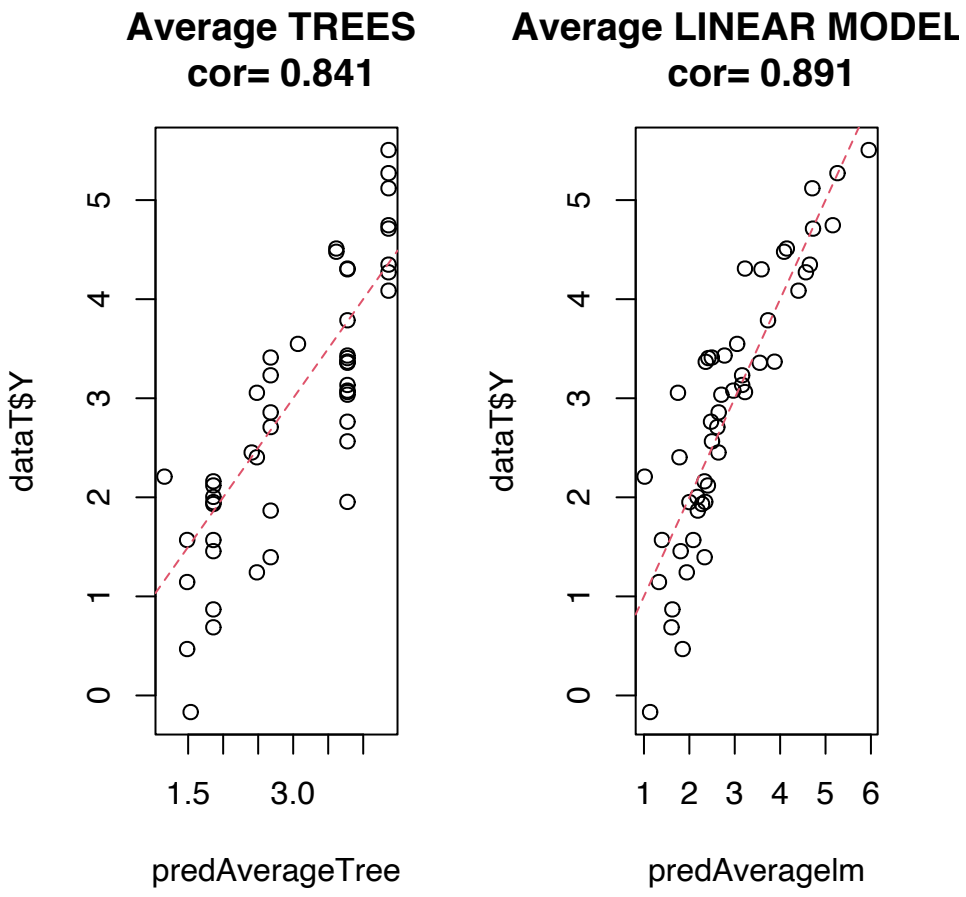
As we can see, our two trees predict things much more differently between them that the two linear models. If we compare the predictions with the real values observed of Y in the T subset:

```
par(mfrow = c(2, 2))
corA <- cor(dataT$Y, predsA)
plot(dataT$Y ~ predsA, main = paste("TreeA cor=", round(corA, 3)))
abline(0, 1, lty = 2, col = 2)

corB <- cor(dataT$Y, predsB)
plot(dataT$Y ~ predsB, main = paste("TreeB cor=", round(corB, 3)))
abline(0, 1, lty = 2, col = 2)

corAlm <- cor(dataT$Y, predsAlm)
plot(dataT$Y ~ predsAlm, main = paste("lmA cor=", round(corAlm, 3)))
abline(0, 1, lty = 2, col = 2)

corBlm <- cor(dataT$Y, predsBlm)
plot(dataT$Y ~ predsBlm, main = paste("lmB cor=", round(corBlm, 3)))
abline(0, 1, lty = 2, col = 2)
```



All models are relatively good (it was an easy dataset!), but linear models seem better. But, what if we combine their predictions (by doing for example an average)?

```
par(mfrow = c(1, 2))
predAverageTree <- rowMeans(data.frame(predsA, predsB))
corAverageTree <- cor(dataT$Y, predAverageTree)
plot(dataT$Y ~ predAverageTree, main = paste("Average TREES \n cor=",
                                             round(corAverageTree, 3)))
abline(0, 1, lty = 2, col = 2)

predAverageLm <- rowMeans(data.frame(predsAlm, predsBlm))
modAverageLm <- cor(dataT$Y, predAverageLm)
plot(dataT$Y ~ predAverageLm,
     main = paste("Average LINEAR MODELS \n cor=", round(modAverageLm, 3)))
abline(0, 1, lty = 2, col = 2)
```



Surprise! the improvement in the predictive capacity of the trees when we average their predictions is rather big, whereas for linear models is insignificant. In general, to attain precise predictions based on trees we could:

1. Take many subsets (*training sets*) from the population.
2. Build a model with each training set.
3. Average the predictions of the different models.

Of course, we usually do not have such a big amount of data: actually, generally we only have a single dataset. But not all is lost. We can do *bootstrapping*

4.7.3 *Bagging*

The methods called *Bootstrap aggregation* or *bagging* perform something similar to what we have just shown, but in a more rigorous and efficient way. In short, we generate a predefined number of different *bootstrap training sets* and adjust a model (a tree) to each of them; at the end we average all the predictions of the different trees (hundreds or thousands), achieving a much more precise prediction than those of individual trees. In general, very deep trees are adjusted (without pruning), which implies that each tree has a great variance (they are overfitting). However, when taken together, the variance is reduced and the prediction accuracy is higher.

In each step of the *bagging* procedure, we take approximately 2/3 of the data (the *training set*) to adjust the corresponding tree. The other third of the data are the *out of bag* (OOB) observations. Each tree is used to predict the values of the OOB observations, so that the predictions are always made for observations that have not been used to train the model (which is a good thing!). Finally, for each observation of the whole dataset we will have a series of predictions (corresponding to the times that each observation has taken part of the OOB subset); these predictions will be averaged to attain a "consensus" prediction.

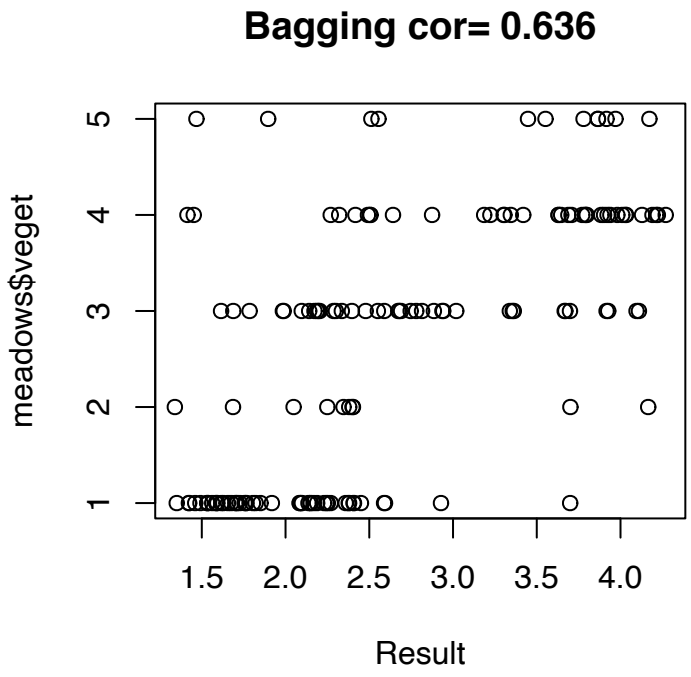
4.7.4 A "homemade" bagging example

Let us go back to our meadows dataset. We will make 5000 trees were we will estimate `veget` as a function of all the traits that we have (`PH`, `TLA`, `SM`, `SLA`, `SLS`, `LH`, `P`, `LS`, `CGO`, `BB.D` and `BB.S`).

```
nreps <- 5000
#Create a matrix to store the predictions of each tree:
predictions <- matrix(
  NA,
  nrow = nrow(meadows),
  ncol = nreps,
  dimnames = list(rownames(meadows), paste("Tree", 1:nreps, sep =
                                           "."))
)

#treeVeget0 <- rpart(veget~ SLA + SM, data=meadows, control = rpart.control(cp=0.05))

for (i in 1:nreps) { #For each repetition
  #Observations in training set by means of bootstrap:
  selected <- sample(1:nrow(meadows), replace = TRUE)
  ##Note: selected has the same number of observation than meadows, but some are
  # repeated and some do not appear (about 1/3). You can check by
  # length(unique(selected)) #Approx. 2/3 of the observations
  OOBAux <- meadows[-selected, ]
  trainingAux <- meadows[selected, ]
  #Lets fit our tree, using trainingAux:
  treeAux <- rpart(veget ~ PH + TLA + SM + SLA + SLS + LH + P + LS + CGO +
                     BB.D + BB.S, data = trainingAux,
                   control = rpart.control(cp = 0.001))
  #Lets predict for the OOB, and store the predictions in the matrix we created before the l
  predictions[-selected, i] <- predict(treeAux, newdata = OOBAux)
  if (i %% 500 == 0) {
    cat(paste("All is good! Rep. number", i, "\n"))
  }
}
```

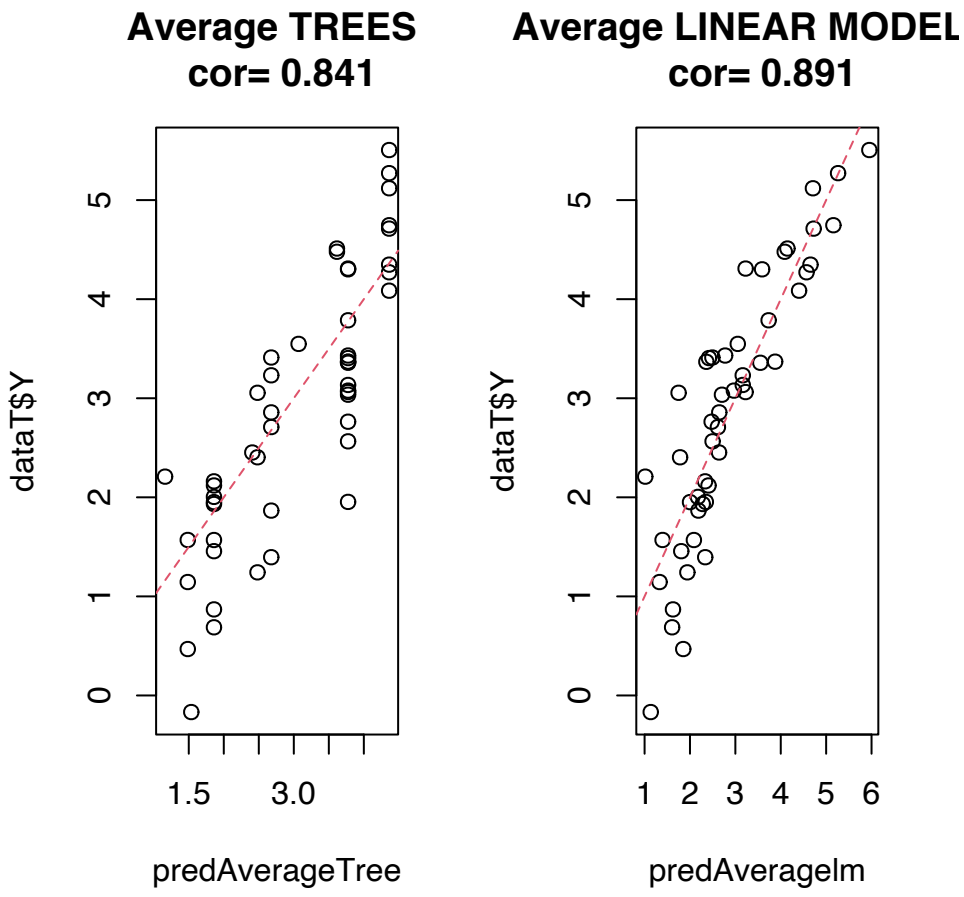
```
## All is good! Rep. number 500
## All is good! Rep. number 1000
## All is good! Rep. number 1500
## All is good! Rep. number 2000
## All is good! Rep. number 2500
## All is good! Rep. number 3000
## All is good! Rep. number 3500
## All is good! Rep. number 4000
## All is good! Rep. number 4500
## All is good! Rep. number 5000
```

```
##Finally, let's average the predictions:
Result <- rowMeans(predictions, na.rm = T)

#And let us examine how much they resemble the real observations:
corBagging <- cor(meadows$veget, Result)
plot(meadows$veget ~ Result,  main = paste("Bagging cor=", round(corBagging, 3)))
```



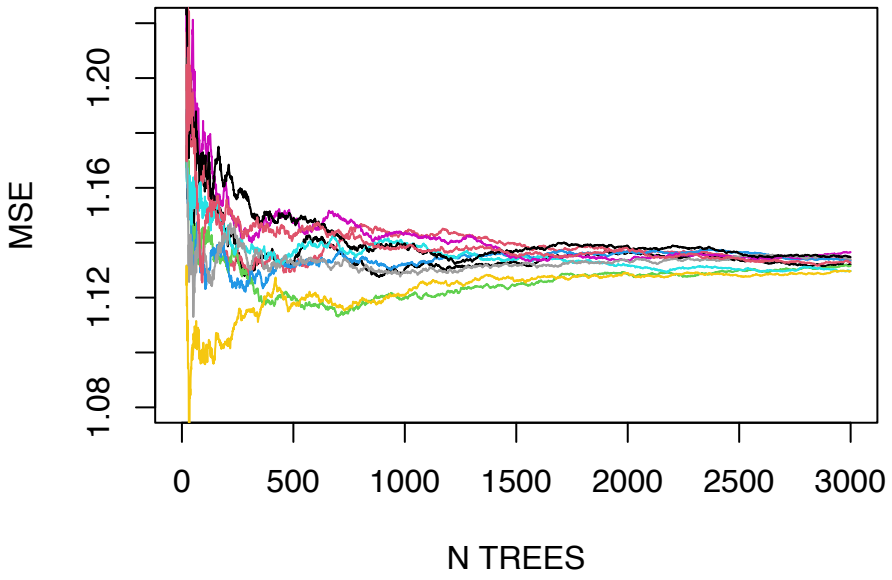
We got a pretty decent predictive capacity! A correlation of 0.636 means that R^2=0.404496, and keep in mind that this refer to data that was not the one used to fit the model. There are still some questions pending, indeed. The main one being: does our predictive ability increases indefinitely as we add more and more trees? Obviously the answer to this is "no". But, how many trees should we average? One solution for this is to examine how mean squared error (MSE) changes as we add more trees:

```
MSE <- rep(NA, nreps)

for (i in 20:nreps) {
  #Let us start in 20 instead of 1, to be more or less sure that all observations
  #have been part of the OOB sample at least once
  averageAux <- rowMeans(predictions[, 1:i], na.rm = T)
  MSE[i] <- mean((meadows$veget - averageAux) ^ 2)
}

plot(20:nreps, MSE[20:nreps], type = "l")
```



Seems like around 1,000 trees MSE stabilizes. In any case, it is convenient to make sure that our algorithm does not contain too few trees, because that might lead to a high variability in the predicted values. The following code (and figure) shows how MSE varies for 10 simulations like the one implemented above (*Warning*:**executing this will take a while!!**).

```
nProcesses <- 10
nreps <- 3000
plot(0, 0, type = "n", xlim = c(0, nreps), ylim = c(1.08, 1.22), ylab = "MSE",
  xlab = "N TREES")
for (k in 1:nProcesses) {
  predictions <- matrix(NA, nrow = nrow(meadows), ncol = nreps,
                        dimnames = list(rownames(meadows),
                                        paste("A", 1:nreps, sep = ".")))
  for (i in 1:nreps) {
    selected <- sample(1:nrow(meadows), replace = TRUE)
    OOBAux <- meadows[-selected, ]
    trainingAux <- meadows[selected, ]
    treeAux <- rpart(veget ~ PH + TLA + SM + SLA + SLS + LH + P + LS + CGO +
                       BB.D + BB.S, data = trainingAux,
                     control = rpart.control(cp = 0.001))
    predictions[-selected, i] <- predict(treeAux, newdata = OOBAux)
  }
  MSE <- rep(NA, nreps)
  for (i in 20:nreps) {
    averageAux <- rowMeans(predictions[, 1:i], na.rm = T)
    MSE[i] <- mean((meadows$veget - averageAux) ^ 2)
  }
  lines(20:nreps, MSE[20:nreps], col = k)
}
```



Which suggests that some value around 1,500 trees is generally safe (although there is still some variability between rounds after 3,000 trees!

An important point to consider is that regression trees allow to have missing values in the predictors without completely losing our observation. This is *very* frequent in trait data, and none of the solutions to this (like data imputation or choosing just those traits for which we have full information and discarding the rest) is fully satisfactory. This indeed happens in our data set:

```
traitColumns <-c("PH", "TLA", "SM", "SLA", "SLS", "LH", "P", "LS", "CGO",  "BB.D", "BB.S")
sum(rowSums(apply(meadows[, traitColumns], 2, is.na))==0)
```

```
## [1] 55
```

Where only 55 observations (i.e. species) have complete trait data.

Random forest models use a more sophisticated version of *bagging*, but we will not continue with them since the main goal here is to explain *boosted regression trees* and to follow the example and approach proposed in Pistón et al. (2019).

4.7.5 Boosted regression trees (BRT)

boosting is another technique that can be applied to improve the predictive ability of regression tree models (although it can actually be applied to other kinds of models). Unlike bagging, boosting is *sequential*. This means that trees are being fitted to the *training data* in an iterative fashion, so that in each step the main focus is on predicting the observations that were not well predicted by the previous trees. In order to do this, boosting algorithms try to optimize (minimize) a loss function (such as deviance), choosing in each step the tree that most reduces that function. This sounds complicated, but conceptually is similar to the minimization of MSE we did with single trees. Like GLM's, BRT can be applied to many types of response variable (binomial, poisson, gaussian…).

Another difference with bagging is that the trees that are used in boosting are usually small ones. We do not want that each individual tree adjusts very well to the particularity of the data, but rather that our procedure *learns* slowly. The process follows these steps:

0. Before anything, we need to choose a series of parameters (frequently called *hyper-parameters*). These include: *1)* tree complexity, which indicates the maximum number of nodes per tree. This somehow reflects the complexity of interactions among variables that we are going to allow –a tree with a single decision does not permit any interaction between variables, whereas when we have a deeper tree partition rules for one predictor can depend on the values of other predictors. *2)* learning rate, which is a scalar that multiplies the contribution of each tree. Explanations about the deeper meaning of this are beyond the scope of this document, but suffices to say that if learning rate is too high, our model will take a shorter time to adjust, but the results are likely to be sub optimal. *3)* bag fraction, which is the proportion of data that are going to be used to train our model (the rest are the out of bag fraction, OOB, that as we have seen are used to validate our model). *4)* the number of trees to adjust. In general, it is recommended to aim for a minimum of 1,000 trees. These parameters must be chosen by the researcher, and they can affect the results, so that careful examination of different combinations of them are recommended. There are also some heuristics, such as aiming for at least 1,000 trees, as we mentioned.

1. The first tree that is fitted is the one which, for the selected value of tree complexity, reduces the deviance (the loss function) by a greatest amount.

2. The next tree, instead of being fitted to the original data, is adjusted to the *residuals of the first tree*. This way, part of the variance that remained unexplained by the first tree will be explained. At this step, predictions for the observations are made taking both trees into consideration.

3. In each of the following steps, a new tree is fitted to the residuals of the combination of previous trees. The optimal number of trees is decided using k-fold cross validation; this basically means that the residual deviance and its standard error are estimated in 10 subsets of the data for different tree numbers, and at the end of the process we (well, rather the algorithm) choose the number of trees that minimizes residual deviance.

4. The final model is a combination of many trees (between hundreds and thousands). The contribution of each individual tree to the final model is reduced by means of the learning rate, which allows the learning process to descend slowly along the surface of the loss function (which is something good: we want our model to learn slowly, but safely).

   Note: each of the trees is adjusted using a random part of the data (the *bag fraction*), so that the process is stochastic. This means that two different BRT's fitted to the same dataset with the same parameters will not yield the same exact result (unless we have chosen a bag fraction of 1, or that we have fixed the randomness of the process –with `set.seed()`). This stochasticity allows to reduce the variance in the final model, increasing the predictive ability of the model.

5. The values adjusted at the end of the process are the sum of all the trees multiplied by the learning rate.

There are a few R packages that can adjust BRT. We are going to use `dismo`, although `gbm` can be rather useful and `caret` has interesting functionalities for selecting hyper-parameters. We have already loaded these packages at the beginning of this R material.

4.7.6 Boosted regression trees for studying the relationship between traits and vegetative reproduction

In the main text of the Pistón et al. (2019) paper we show how to use BRT's to estimate the effects of different traits on seed and vegetative reproduction in several habitats. Here, we will reproduce these analyses, focusing on vegetative reproduction for meadows. We used the function `gbm.step`

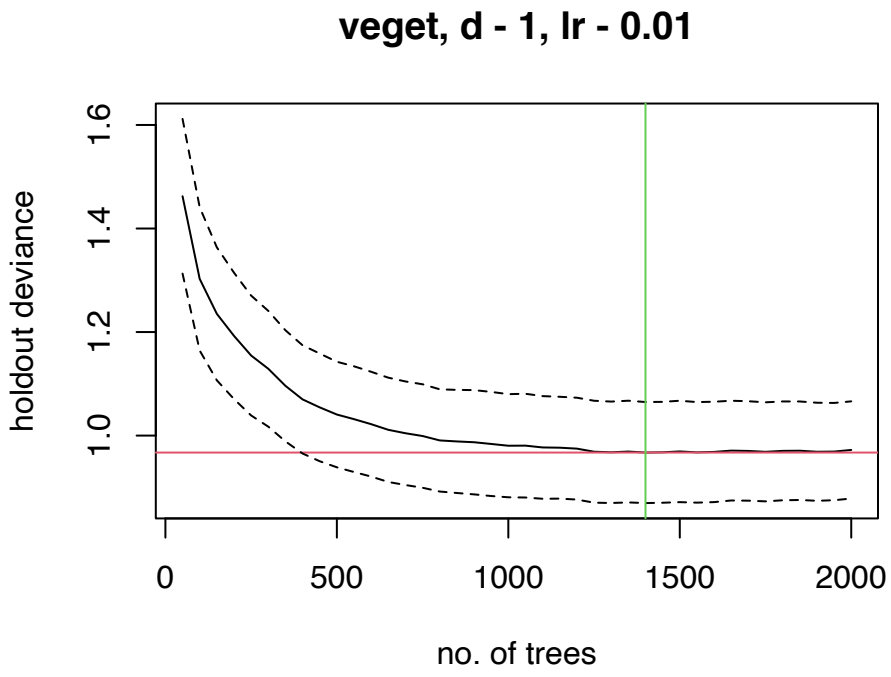
The first thing to do is to select the hyper-parameters of our BRT models. We chose a bag fraction of 75% because we wanted to use a bigger proportion of the data to fit the model and less observations to test it. Then, we had to choose an adequate learning rate. It is generally proposed that you keep a learning rate value that results in fitting at least 1,000 trees. This can be done like this (please check the help of the function if you have any doubts about other parameters):

```
BRT.meadows.LR0.1 <- gbm.step(data = meadows,
                              gbm.x = c(3:13), #columns of predictors
                              gbm.y = 2, # Column of the response variable
                              family = "gaussian",
                              tree.complexity = 1,
                              learning.rate = 0.1,
                              bag.fraction = 0.75,
                              n.trees = 50,
                              verbose = FALSE)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for veget and using a family of gaussian
## Using 148 observations and 11 predictors
## creating 10 initial models of 50 trees
##
##  folds are unstratified
## total mean deviance =  1.8948
## tolerance is fixed at  0.0019
## now adding trees...
## restart model with a smaller learning rate or smaller step size...
```

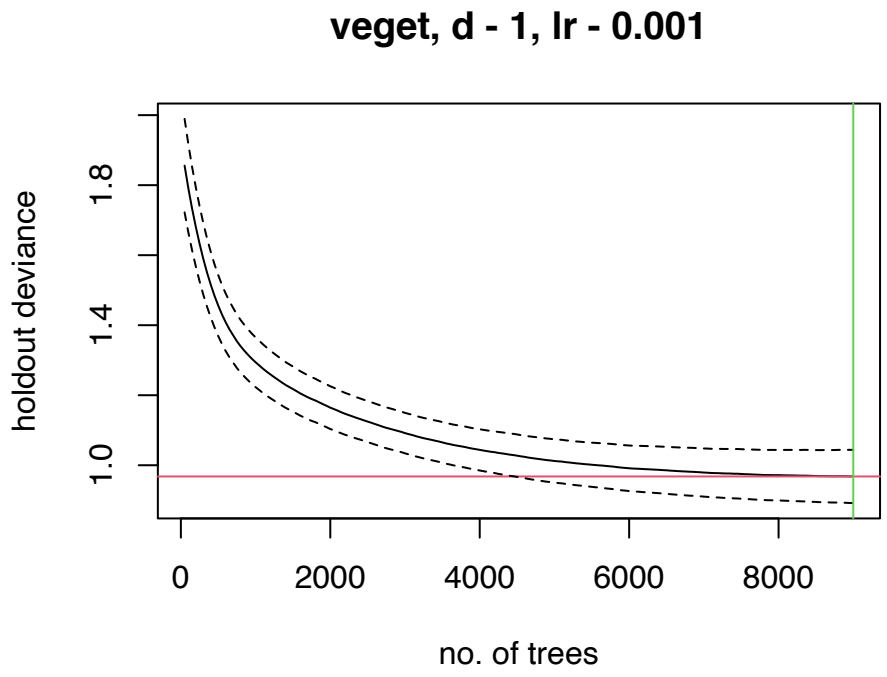
```
BRT.meadows.LR0.01 <- gbm.step(data = meadows,
                               gbm.x = c(3:13), #columns of predictors
                               gbm.y = 2, # Column of the response variable
                               family = "gaussian",
                               tree.complexity = 1,
                               learning.rate = 0.01,
                               bag.fraction = 0.75,
                               n.trees = 50,
                               verbose = FALSE)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for veget and using a family of gaussian
## Using 148 observations and 11 predictors
## creating 10 initial models of 50 trees
##
##  folds are unstratified
## total mean deviance =  1.8948
## tolerance is fixed at  0.0019
## now adding trees...
```




```
BRT.meadows.LR0.001 <- gbm.step(data = meadows,
                                gbm.x = c(3:13), #columns of predictors
                                gbm.y = 2, # Column of the response variable
                                family = "gaussian",
                                tree.complexity = 1,
                                learning.rate = 0.001,
                                bag.fraction = 0.75,
                                n.trees = 50,
                                verbose = FALSE)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for veget and using a family of gaussian
## Using 148 observations and 11 predictors
## creating 10 initial models of 50 trees
##
##  folds are unstratified
## total mean deviance =  1.8948
## tolerance is fixed at  0.0019
## now adding trees...
```

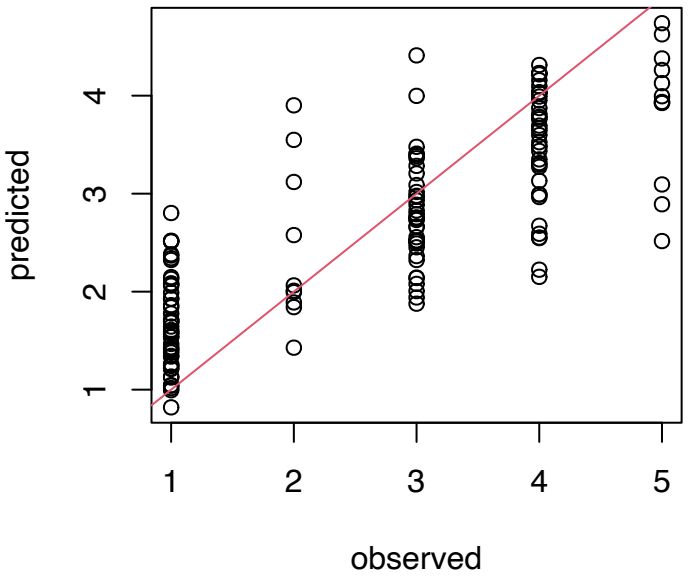


By default, the `gbm.step` function gives a very detailed feedback on what is happening, but to not overflow the output here we set the argument `verbose` to `FALSE` to only display some info. You can also completely silence is by setting `silent` to `TRUE`. The plots accompanying each function show us very well what the function is doing: as more trees are added, the residual variance in the testing fraction of the data is evaluated, and the function selects the optimum number of trees (when the deviance minimizes). This strategy implies that our model is extracting as much information from our data as possible, while avoiding overfitting. In the case of the last BRT we fitted, we ended up with `BRT.meadows.LR0.001$n.trees` = 9000 trees.

In our case, we will select a tree complexity of 0.001. Choosing smaller learning rates does not hurt, but calculations take longer, and you should be aware that perhaps you have to increase the value of the argument `max.trees` (set at 10,000 by default) if you don't want your model to stop before reaching the optimum.

So, how can we evaluate our model? We can start by plotting predicted versus observed values, and even estimate an R^2 value using the ratio of residual to null deviance:

```
predicted <- predict(BRT.meadows.LR0.001, n.trees = BRT.meadows.LR0.001$n.trees)
observed <- meadows$veget
plot(predicted ~ observed)
abline(0, 1, col = 2)
```

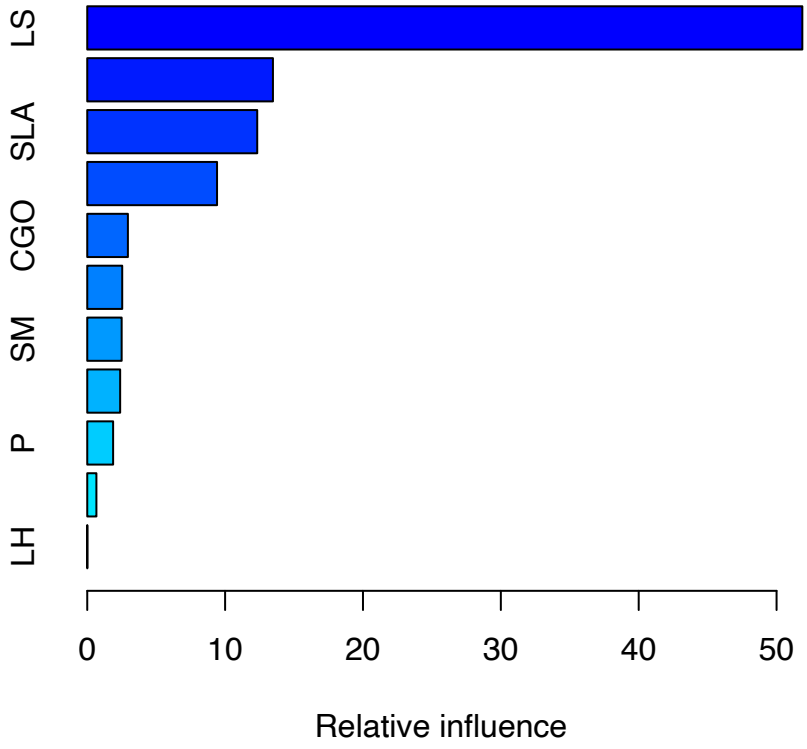



```
R2BRT.meadows <- 1 - (BRT.meadows.LR0.001$self.statistics$mean.resid /
      BRT.meadows.LR0.001$self.statistics$mean.null)
R2BRT.meadows
```

```
## [1] 0.6430943
```

Not bad! 64.3% of the variability in our data. Now, we may be interested in which predictors are more important. For this we can simply do this:

```
summary(BRT.meadows.LR0.001)
```




```
##       var      rel.inf
## LS     LS 51.870130155
```

```
## BB.S BB.S 13.469617036
## SLA   SLA 12.329581020
## BB.D BB.D  9.414826191
## CGO   CGO  2.948833351
## SLS   SLS  2.536779914
## SM     SM  2.496301853
## TLA   TLA  2.387314831
## P       P  1.876756763
## PH     PH  0.664552035
## LH     LH  0.005306851
```

This yields a plot and a table showing a ranking of our predictors according to their relative importance. It seems that `LS` is by far the most important trait in determining vegetative reproduction in this dataset, whereas `BB.S` and `SLA` appear in a secondary position. `LH` seems to have no influence whatsoever. Note that this is not our final model though; we still need to fine-tune some things.

Once we have a learning rate, we have to choose tree complexity. In our case, this was one of the main questions we were interested in in the paper: how complex are interactions between traits as determinants of vegetative and seed reproductions? To do this, we basically performed a sensitivity analyses, where we fitted 100 models for each `tree.complexity` value between 1 (all individual trees have a single partition) and 15 (many partitions are allowed in each tree, allowing for very complex interactions). For each of these trees, we estimated its R^2 using the ratio of residual to null deviance. We also stored the importance of the predictors in each repetition. Because it takes a long while to compute this for 100 models for each tree complexity level, we will simply use 10 in this example, and only for tree complexities from 1 to 10 (spoiler: we found more than that does not improve predictive ability). Feel free to do it as in the paper, if you feel curious and have some free hours. Depending on the computer that you use, even this shortened version can take some time. So if you don't want to wait at all, we also give you the option to load the output of the next chunk of R code, in which case you of course don't run this bit, but the line that loads the result from an Rdata file.

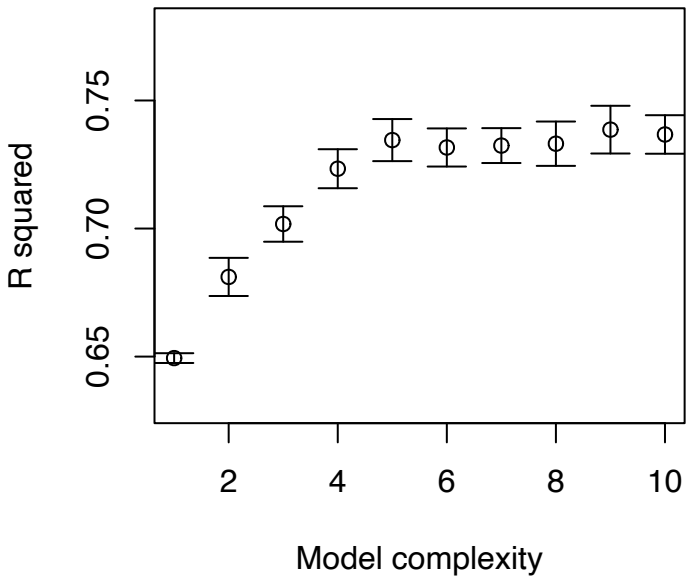
```
R2Obs.meadows <- list()
importancePred.meadows <- list()
nreps <- 10 #number of simulations
for (tcomp in 1:10) {
  R2Obs.meadows[[tcomp]] <- numeric(nreps)
  importancePred.meadows[[tcomp]] <- data.frame(matrix(NA, nrow = length(3:13),
                                                       ncol = nreps))
  for(i in 1:nreps){
    if (i == 1) {
      cat(paste("Starting tc =", tcomp, "\n"))
    }
    BRT.meadows <- gbm.step(data=meadows,
                            gbm.x = c(3:13),
                            gbm.y = 2,
                            family = "gaussian",
                            tree.complexity = tcomp,
                            learning.rate = 0.001,
                            bag.fraction = 0.75,
                            n.trees = 50,
                            silent = T, plot.main=F, plot.folds=F
                            )

    #R2 adj:
    R2Obs.meadows[[tcomp]][i] <- 1 - (BRT.meadows$self.statistics$mean.resid /
             BRT.meadows$self.statistics$mean.null)
```

```
    if (i == 1) {
      rownames(importancePred.meadows[[tcomp]]) <- sort(rownames(summary(BRT.meadows)))
    }
    importancePred.meadows[[tcomp]][, i] <-
      summary(BRT.meadows)[rownames(importancePred.meadows[[tcomp]]), ]$rel.inf
  }
}
```

```
load(here::here("data", "chapter4", "BRT.Rdata"))
```

We then examined how R^2 improves as tree complexity increases:

```
means <- sapply(R2Obs.meadows, mean)
sds <- sapply(R2Obs.meadows, sd)
plot(1:length(R2Obs.meadows), means, ylim = c(0.63, 0.78), ylab = "R squared",
     xlab = "Model complexity")
for (i in 1:length(R2Obs.meadows)) {
  arrows(x0 = i, x1 = i, y0 = means[i] - sds[i], y1 = means[i] + sds[i],
    angle = 90, code = 3, length = 0.1)
}
```

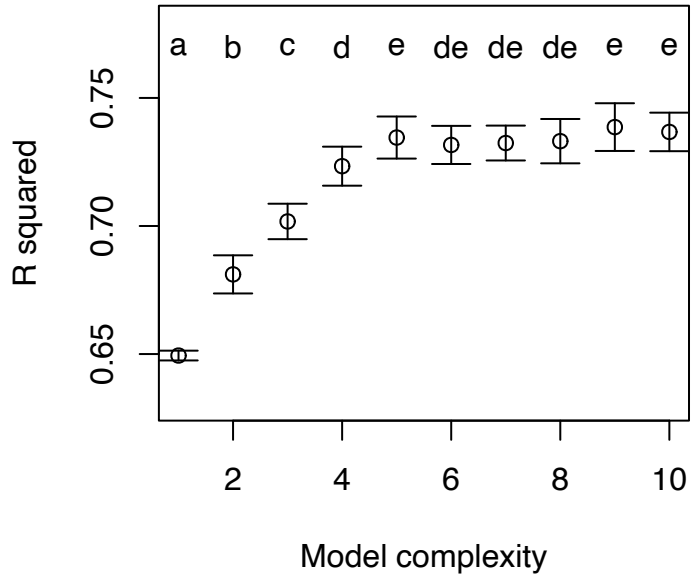


Indeed, R^2 only increases as tree complexity increases. In order to select that lowest tc value possible, but not too low, we performed an ANOVA analysis where all the estimated R^2 where the response variable and the tree complexity (treated as a factor), was the explanatory variable. We then performed a Tukey test and selected the first tc value that did not differ significantly from the last one (indicating that model performance had reached a plateau):

```
nreps <- 10
tcFactor <- as.factor(rep(1:10, each = nreps))
R2Vector <- unlist(R2Obs.meadows)
model <- lm(R2Vector ~ tcFactor)
library(multcomp)

TukeyModel <- glht(model, linfct = mcp(tcFactor = "Tukey"))
TukeyLetters <- cld(TukeyModel)$mcletters$Letters
```

```
## Warning in RET$pfunction("adjusted", ...): Completion with error > abseps

## Warning in RET$pfunction("adjusted", ...): Completion with error > abseps

## Warning in RET$pfunction("adjusted", ...): Completion with error > abseps

## Warning in RET$pfunction("adjusted", ...): Completion with error > abseps
```

```
plot(1:length(R2Obs.meadows), means, ylim=c(0.63, 0.78), ylab = "R squared",
     xlab = "Model complexity")
for (i in 1:length(R2Obs.meadows)){
  arrows(x0 = i, x1 = i, y0 = means[i] - sds[i], y1 = means[i] + sds[i],
         angle = 90, code = 3, length = 0.1)
}
text(x = 1:length(R2Obs.meadows), y = 0.77, labels = TukeyLetters)
```



So, in this case we will choose a tree complexity value of 4. Note that this might change, since the process is not completely deterministic. The average R^2 in our case `round(means[5], 3)`. How important are the different predictors across all our simulations in this model?

```
sort(rowMeans(importancePred.meadows[[5]]), decreasing = T)
```

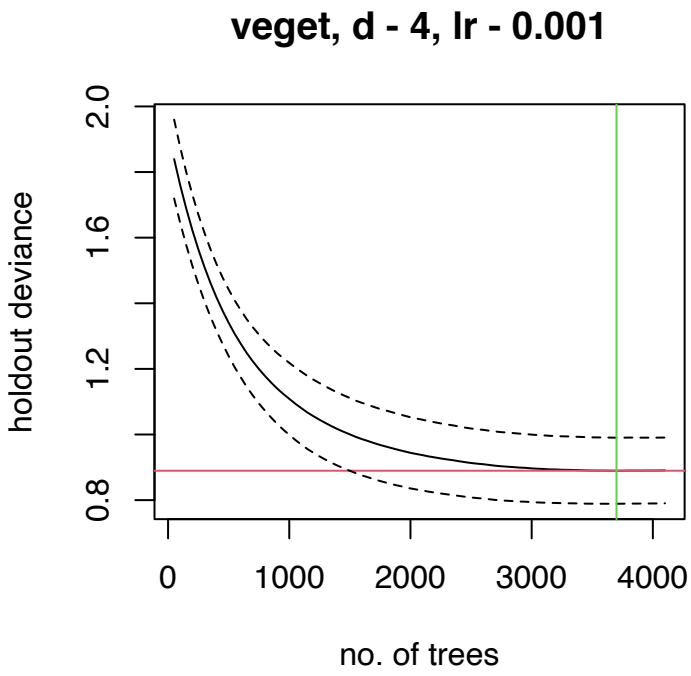
```
##         LS        SLA        SLS       BB.S        TLA       BB.D         PH
## 42.2170549 13.3185466 10.8412292  9.0670267  8.9838632  6.5760523  3.5461251
##         SM        CGO          P         LH
##  1.9140820  1.7357537  1.4499400  0.3503263
```

Still `LS` is the most important predictor, and now `SLA` seems to have climbed a position compared to the first BRT we fitted. We can examine the shape of the response of `veget` to `LS` by means of a partial dependence plot. In this kind of plots all other explanatory variables are "fixed". To make this simpler, let's fit a single BRT with the finally selected parameters, and use it to make the plot:

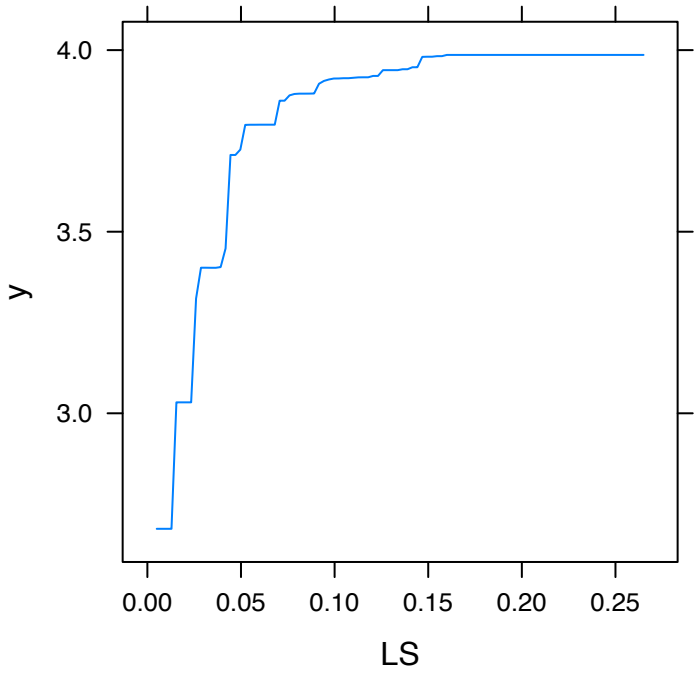
```
BRT.meadows.final <- gbm.step(data = meadows,
                              gbm.x = c(3:13),
                              gbm.y = 2,
                              family = "gaussian",
```

```
                          tree.complexity = 4,
                          learning.rate = 0.001,
                          bag.fraction = 0.75,
                          n.trees = 50,
                          verbose = F)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for veget and using a family of gaussian
## Using 148 observations and 11 predictors
## creating 10 initial models of 50 trees
##
##  folds are unstratified
## total mean deviance =  1.8948
## tolerance is fixed at  0.0019
## now adding trees...

## fitting final gbm model with a fixed number of 3700 trees for veget
```




```
plot.gbm(BRT.meadows.final, i.var = c("LS"))
```



Seems like vegetative reproduction increases as `LS` increases, but instead of doing it linearly, it follows a saturating relationship. Note how easily we have been able to find this kind of response by using BRTs, and how hard it would have been with linear models.

Finally, we investigated how interactions between pairs of traits affected the response variable. The function `gbm.interactions` evaluates this, and returns a list with the relative importance of the interactions between pairs of predictors (see `gbm.interactions` for details about how this is done):

```
gbm.interactions(BRT.meadows.final)$rank.list
```

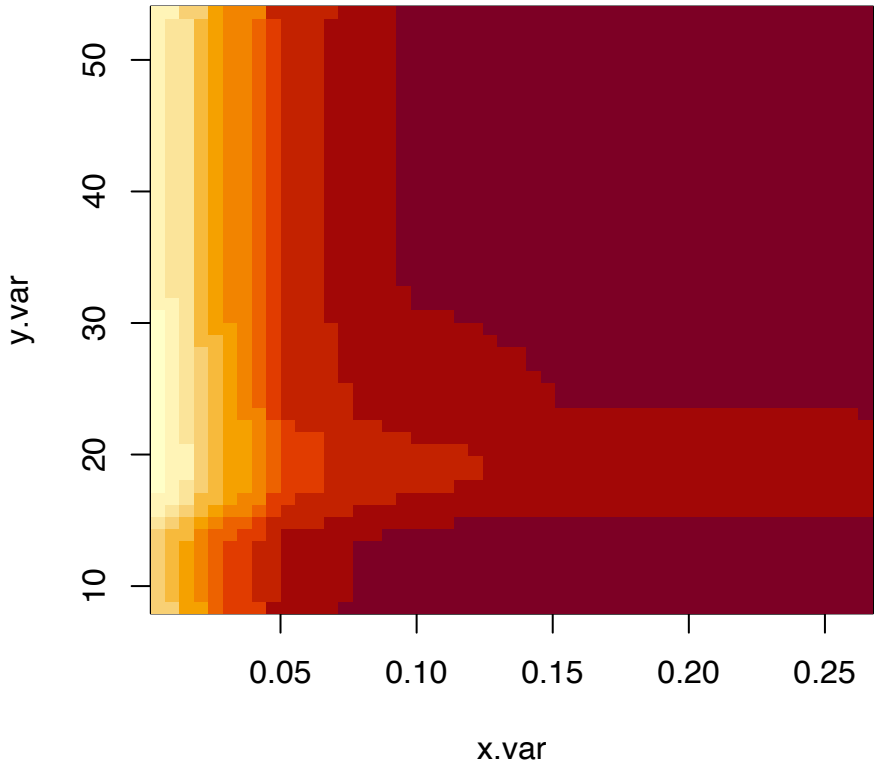
```
## gbm.interactions - version 2.9
## Cross tabulating interactions for gbm model with 11 predictors

## 1 2 3 4 5 6 7 8 9 10

##   var1.index var1.names var2.index var2.names int.size
## 1          8         LS          2        TLA     2.75
## 2          8         LS          4        SLA     2.42
## 3         10       BB.D          4        SLA     0.74
## 4         11       BB.S          8         LS     0.60
## 5         11       BB.S          4        SLA     0.22
## 6         10       BB.D          2        TLA     0.21
```

We can even make a perspective plot to visually examine these interactions. Let us explore the interaction between `LS` and `SLA`:

```
gbm.perspec(BRT.meadows.final,
            8,  # LS
            4, #SLA,
            z.range = c(1.9, 4.15),
            phi = 20, col = "black", cex.axis = 0.8,
            cex.lab = 1, ticktype = "detailed",
            y.label = "SLA",
            x.label = "LS",
```

```
            leg.coords = c(0.5,2.3),
            perspective = FALSE,
            smooth = "average")
```

```
## maximum value = 4.1
```



> Unfortunately, there is a little bug in the `gbm.perspec` function that does not print the labels on the y and x axis of the plot as we requested via the `y.label` and `x.label` arguments. So you will have to imagine that x.var is LS, and y.var is SLA.

This plot reveals that there are two alternative optima for vegetative reproduction in our dataset, which come from the interaction between SLA and LS. Whereas for low LS values vegetative reproduction is always very low, for higher LS values vegetative reproduction maximizes for low and high values of SLA, whereas there is a local minimum (sort of a valley) for SLA values around 20.

5 – Community Weighted Mean (CWM) and Functional Diversity (FD)

This chapter corresponds to Chapter 5 of the reference textbook, in which we deal with characterizing the functional trait structure of biological communities. In particular, we will start focusing on Community Weighted Mean (CWM) and then we will see methods to estimate Functional Diversity (FD).

5.1 Community Weighted Mean (CWM)

In this exercise we will learn how to compute one important index of *community trait structure*, i.e. Community Weighted Mean (*CWM*). This exercise follows the first part of Chapter 5 of the reference textbook, so all theoretical and mathematical issues beyond the indices described in this exercise can be found there.

5.1.1 Data

We will work with first invented data, as used in the Chapter (specially Fig.5.2, with data created below), as well as data available in the help of the function `functcomp`. Then field data from an climatic gradient in NE Spain will be used.

In the *NE Spain data*, 5 vegetation belts along a climatic and altitudinal gradients were sampled. The data is based on two publications de Bello et al. (2006; 2009). The gradient characterized by the *5 vegetation belts* is going from a shrubland in Monegros, in the Aragon region (basically a desert with rainfall around 320 mm per year and altitude around 200 m a.s.l) to a subalpine meadow in the Catalan Pirinees (rainfall slightly below 1000 mm per year and altitude around 2000 m a.s.l). We characterized each belt by a *moisture index*, which is rainfall divided by potential evapotranspiration (just google UNEP aridity index). In each vegetation belt a total of 12 plots were surveyed (10x10m plots divided into 100 $1m^2$ quadrats). Thus for each plot species frequency over the 100 quadrats is given in the "speciesXplotsNE.txt" file (*notice* that in the file species are rows and plots are columns, while some authors and R function prefer the data in a *transposed* form, with species information using columns and plots using rows; we will start with this format, shown in Chapter 5 and explain why in some cases it is better to transpose the data). Out of the 12 plots per vegetation belt, 4 had high grazing pressure, 4 intermediate and in 4 grazing was abandoned. Information about plot location along the 5 vegetation belts are included in the file "environXplotsNE.txt" (column "vegbelt", 1 is the driest and lowest vegetation belt, 5 is the highest and wettest vegetation belt, see also "moisture index"; in column "grazing" 0=abandonment, 1=intermediate grazing, 2=intensive grazing). Traits information is included in "speciesXtraitsNE.text", similar to the data used in Chapter 3, see further below. Here is the NE Spain data:

```
spxp <- read.table(here::here("data", "chapter5", "speciesXplotsNE.txt"), row.names = 1, hea
spxt <- read.table(here::here("data", "chapter5", "speciesXtraitsNE.txt"), row.names = 1, he
envxp <- read.table(here::here("data", "chapter5", "environXplotsNE.txt"), row.names = 1, he
```

Notice that for all files we specifically asked that the names of species and plots are not part of the matrix, but they are used (in this case) as names of the rows (`row.names=1`) and as

column names (`header=T`). Be careful to use these options carefully when you consider your data, because they could be in a different arrangement.

5.1.2 Required packages and additional functions

We also need to load various packages in order to run the analyses. Please ensure that you have installed these packages in your R library before you load them.

```
library(FD)
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.0.2
```

5.1.3 Calculation of CWM with invented data

Let's start by recreating the example at Fig. 5.2 in the reference book. To do this let's create first some 'species x community' matrix and then the 'species x traits' matrix. *Notice* that we will start computing everything by hand, to learn the overall process, although existing functions in R can be used instead much more simply (see later in the section).

```
comm1 <- c(10, 10, 10, 10, 10, 0, 0)
comm2 <- c(49, 0, 5, 0, 5, 0, 11)
comm3 <- c(0, 6, 0, 4, 0, 2, 8)
spxcom.matrix <- cbind(comm1, comm2, comm3)
rownames(spxcom.matrix) <- paste("species", 1:7)
spxcom.matrix
```

```
##           comm1 comm2 comm3
## species 1    10    49     0
## species 2    10     0     6
## species 3    10     5     0
## species 4    10     0     4
## species 5    10     5     0
## species 6     0     0     2
## species 7     0    11     8
```

```
Bodysize <- c(10, 20, 30, 40, 50, NA, 70)
Carnivory <- c(1, 1, 0, 1, 0, 1, 0)
spxtraits.matrix <- cbind(Bodysize, Carnivory)
rownames(spxtraits.matrix) <- paste("species", 1:7)
spxtraits.matrix
```

```
##           Bodysize Carnivory
## species 1       10         1
## species 2       20         1
## species 3       30         0
## species 4       40         1
## species 5       50         0
## species 6       NA         1
## species 7       70         0
```

As you can see this data reflects an hypothetical case of a dataset with a total of 3 communities, containing a total of 7 species, for which we have data for 2 traits, body size and whether species are carnivourous (1=yes) or not (0=no). Species 6 has missing data for body size.

To compute CWM we first need the relative abundance of the species in each community. For this we need first to compute the total abundance in each community for example as:

```
totalabb <- colSums(spxcom.matrix)
totalabb
```

```
## comm1 comm2 comm3
##    50    70    20
```

Then, we need to divide the abundance of each species in each plot by the total abundance of the plot. In R it could be done as following (*notice* that, because of the way R handle matrix operation the matrix needs to be transposed in this case; notice also that the sum of the relative abundances within a plot should be equal to 1, as we show):

```
sp.rel.abb <- t(spxcom.matrix) / totalabb #you need to transpose the matrix here
t(sp.rel.abb) #we just transpose the matrix "back", to display it as in the book
```

```
##           comm1      comm2 comm3
## species 1   0.2 0.70000000   0.0
## species 2   0.2 0.00000000   0.3
## species 3   0.2 0.07142857   0.0
## species 4   0.2 0.00000000   0.2
## species 5   0.2 0.07142857   0.0
## species 6   0.0 0.00000000   0.1
## species 7   0.0 0.15714286   0.4
```

```
colSums(t(sp.rel.abb)) #this is the total sum of relative abundances in each plot
```

```
## comm1 comm2 comm3
##     1     1     1
```

As we discuss it in the book (Chapter 5 for example), because the *presence of a missing value (NA)* in the trait matrix we will have some *problems*. In particular species 6 is present in community 3, with a relative abundance of 0.1 (i.e. 10%). Because we do not have data about body size for this species, we need to remove the species from community 3 and compute new relative abundances (because the sum of relative abundance still needs to be sum 1). To do this, for example, we can:

```
spxcom.matrix.nosp6 <- spxcom.matrix #create a copy of the species x community matrix
spxcom.matrix.nosp6["species 6",] <-  0 #put zero for species 6 in all communities
sp.rel.abb.nosp6 <- t(spxcom.matrix.nosp6) / colSums(spxcom.matrix.nosp6)
t(sp.rel.abb.nosp6)
```

```
##           comm1      comm2     comm3
## species 1   0.2 0.70000000 0.0000000
## species 2   0.2 0.00000000 0.3333333
## species 3   0.2 0.07142857 0.0000000
## species 4   0.2 0.00000000 0.2222222
## species 5   0.2 0.07142857 0.0000000
## species 6   0.0 0.00000000 0.0000000
## species 7   0.0 0.15714286 0.4444444
```

As you can see the relative abundances in community 3 have slightly changed. This means now that, in principle, for each trait we should use a different plot composition data which is of course complicated. *Notice* that we could have also removed the species from the whole matrix, for example by doing `spxcom.matrix.nosp6<-spxcom.matrix[-6, ]` but this would have made the 'species x community' matrix smaller than the 'species x traits' matrix, which is definitely NOT a good way forward. For example in this case the species has other traits information so that we can still use it for computing trait dissimilarity (see R material Ch 3) and therefore functional diversity (R material Ch 5.2. In some cases, indeed it might be quicker to remove the species from all matrices, see below.

We can now compute the CWM for the two traits. For Carnivory, where we do not have NAs, we can use the full data, with all species ('sp.rel.abb'), while for Body size we need the version of the plot data without species 6 ('sp.rel.abb.nosp6'). The CWM is computed, for each plot (community), by 1) multiplying the trait values by species relative abundance and 2) summing the resulting values. This is done in the following way, first for Carnivory:

```
t(sp.rel.abb) * spxtraits.matrix[, "Carnivory"]
```

```
##           comm1 comm2 comm3
## species 1   0.2   0.7   0.0
## species 2   0.2   0.0   0.3
## species 3   0.0   0.0   0.0
## species 4   0.2   0.0   0.2
## species 5   0.0   0.0   0.0
## species 6   0.0   0.0   0.1
## species 7   0.0   0.0   0.0
```

```
colSums(t(sp.rel.abb) * spxtraits.matrix[, "Carnivory"])
```

```
## comm1 comm2 comm3
##   0.6   0.7   0.6
```

This means that in comm1 and comm3, a 60% of the total abundance is composed by carnivourous organisms. In comm2, it is 70%.

Before we continue, a word of caution. Here, and below, *notice* that we use the expression "60% of the total abundance is composed by carnivourous *organisms*" and NOT "*species*". This is because, using relative abundance, the CWM indicates which proportion of the total abundance in a plot is composed by organisms of a given type, for example how many individuals in the `comm1` are carnivorous, irrespectively of the proportion of species. Of course, in `comm1` all species have the same abundance (10 for all species, so that in that case we can say that 60% of the species are carnivorous). But in `comm2` the abundance is quite uneven across species, so that 70% indicates the proportion of organisms, and not species, that in the plot are carnivourous.

Having said so, we can compute the same for Body size:

```
t(sp.rel.abb.nosp6) * spxtraits.matrix[, "Bodysize"]
```

```
##           comm1     comm2     comm3
## species 1     2  7.000000  0.000000
## species 2     4  0.000000  6.666667
## species 3     6  2.142857  0.000000
## species 4     8  0.000000  8.888889
## species 5    10  3.571429  0.000000
## species 6    NA        NA        NA
## species 7     0 11.000000 31.111111
```

```
round(colSums(t(sp.rel.abb.nosp6) * spxtraits.matrix[, "Bodysize"],
              na.rm = T), 1) #you need to add the argument 'na.rm=T'; here we used 'round' j
```

```
## comm1 comm2 comm3
##  30.0  23.7  46.7
```

Which means that the weighted average of body size is the biggest in comm3 and the smallest in comm2.

Of course you do not need to compute CWM always by hand! There are existing functions (such as `functcomp` in the package `FD`) that does all the things we did above directly for you. This was simply an example to show the functioning of the underlying processes. We can get (basically) the same results (but see below) by doing the following:

```
library(FD)
functcomp(spxtraits.matrix, t(spxcom.matrix)) #do not forget to transpose the species x comm
```

```
##       Bodysize Carnivory
## comm1 30.00000         1
## comm2 23.71429         1
## comm3 46.66667         1
```

As you see in the script above, the *function 'functcomp'* does all the stepts we did above, by 'hand', in just one line. How convenient! The function needs a 'species x traits' matrix (more specifically a `data.frame`), as first argument, and as second argument we need to provide the 'species x community' matrix, which has a form where species are columns and plots as rows (that is why we needed to transpose it!). It is *important to notice* that these two objects need to be of the same size (i.e. same number of species) and that the species names should be *EXACTLY* the same in both, even the same order. This is very often a problem we find when we are working with students. So be careful! We suggest checking these things before doing the calculations. For example, for small datasets this can be checked easily (other solution are needed for bigger datasets of course):

```
rownames(spxtraits.matrix) == rownames(spxcom.matrix)
```

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

Notice also that in the *results* we obtained above *there is something apparently strange*! they are not the same we got by hand, so far. Specifically, we got the same results obtained above for Body size, but not for Carnivory. For this trait, using the function `functcomp` we got, as a result, the value 1 for all 3 plots. What does this mean? This apparently counterintuitive result is simply because the function 'functcomp', in the case of these *binary traits, or any categorial trait*, does not compute by default the CWM, but it rather tells us what is the most dominant type in each plot. In practice the function tells us that in all the 3 plots considered, the dominant type is 'carnivorous' (remember that in all communities there was more than 50% of carnivorous organisms). To obtain the proportion of each type, which is what we computed above using `colSums(t(sp.rel.abb)*spxtraits.matrix[, "Carnivory"])`, we can just write:

```
functcomp(spxtraits.matrix, t(spxcom.matrix), CWM.type = "all")
```

```
##       Bodysize Carnivory_0 Carnivory_1
## comm1 30.00000         0.4         0.6
## comm2 23.71429         0.3         0.7
## comm3 46.66667         0.4         0.6
```

With the argument `CWM.type = "all"` The function `functcomp` now gives us the CWM for both carnivourous and non-carnivorous categories. This shows that in the first community is composed by a total of 40% non-carnivorous organisms, and consequently, a 60% of carnivorous organisms. Obvioulsy one would expect that the sum of these two values would be 1, so that the columns 'Carnivory_0' and 'Carnivory_1' are "complementary". This is the case in this example, and it should be always like that! But....sometimes 'functcomp' gets into *troubles in the presence of NAs*. Let's see the example 2, from the help of `functcomp` (i.e. `?functcomp`). The example 2 uses the data already available in the package. The species x trait matrix and the species x community matrix are respectively:

```
dummy$trait
```

```
##     num1 num2 fac1 fac2 ord1 ord2 bin1 bin2
## sp1  9.0  4.5    A    X    3    2    0    1
## sp2  8.1  6.0    A    Z <NA>    1    0    1
## sp3   NA  2.3    C    Y    5    3    1    1
## sp4  3.2  5.4    B    Z    1    7    0    0
## sp5  5.8  1.2    C    X    2    6   NA    0
## sp6  3.4  8.5    C    Y    2    1    1    1
## sp7  7.5  2.1    B    X    3    2    1    0
## sp8  4.3  6.5 <NA>    Z    1    3    0    1
```

```
t(dummy$abun)
```

```
##     com1 com2 com3 com4 com5 com6 com7 com8 com9 com10
## sp1    1    0    2    1    0    0    3    0    4     0
## sp2    1    0    0    0    0    3    5    0    1     4
## sp3    0    0    0    7    2    0    0    0    1     1
## sp4    0    2    0    0    3    0    3    0    3     0
## sp5    4    1    0    0    3    5    0    6    0     0
## sp6    2    0    1    0    0    6    0    2    0     0
## sp7    0    0    0    0    0    1    0    1    2     6
## sp8    0    5    3    0    0    6    0    2    0     1
```

Let's now consider the first 3 traits, and particularly the 3rd one, called 'fac1', which is a categorical trait with 3 levels (A, B and C). For this trait, one species (sp8) has a missing trait value. Species 8, was present in com2, com3, com6, com8 and com10. Let's see what happens in the calculations:

```
ex2 <- functcomp(dummy$trait[, 1:3], dummy$abun, CWM.type = "all")
# we just consider the first 3 traits for this example
ex2
```

```
##           num1     num2    fac1_A     fac1_B     fac1_C
## com1  5.887500 4.037500 0.2500000 0.00000000 0.75000000
## com2  4.212500 5.562500 0.0000000 0.25000000 0.12500000
## com3  5.716667 6.166667 0.3333333 0.00000000 0.16666667
## com4  9.000000 2.575000 0.1250000 0.00000000 0.87500000
## com5  4.500000 3.050000 0.0000000 0.37500000 0.62500000
## com6  5.095238 5.528571 0.1428571 0.04761905 0.52380952
## com7  7.009091 5.427273 0.7272727 0.27272727 0.00000000
## com8  5.245455 3.572727 0.0000000 0.09090909 0.72727273
## com9  6.870000 4.245455 0.4545455 0.45454545 0.09090909
## com10 7.427273 3.783333 0.3333333 0.50000000 0.08333333
```

As we mentioned, when using `CWM.type = "all"`, the function `functcomp` will provide the proportion of each of the levels within a categorical trait. So, in this case, with the 3 levels of 'fac1', it provides 3 columns. *The sum of these 3 colums should equal to 1!* but they don't.

```
rowSums(ex2[, 3:5])
```

```
##      com1      com2      com3      com4      com5      com6      com7      com8
## 1.0000000 0.3750000 0.5000000 1.0000000 1.0000000 0.7142857 1.0000000 0.8181818
##      com9     com10
## 1.0000000 0.9166667
```

For example, in com2 there is 0% of type A, 25% of type B and 12.5% of the type C. What happened with the remaining 62.5% of the total abundance? Species (sp8) had an abundance of 5 in com2, where the total abundance was 2+1+5=8. As a matter of fact 5/8=0.625, so the missing 62.5% is exactly due to species 8. As you can see the total is not 1 for all plots in which there was a species having a missing value (i.e. sp8 in com2, com3, com6, com8 and com10).

We can understand why the function `functcomp` does this, i.e. because it does not want to assume the trait values for the missing species. At the same time, for other calculations (for example for quantitative traits), the `functcomp` simply removes the species with NAs from the plots where the species is present, as shown above when removing species 6 in the invented data. So, in the case of quantitative traits, the approach of the function is different, and users should be aware of this. Let's consider the first trait in the 'dummy$trait' data (num1) for which 'sp3' has a missing value. This species is present, for example, in com5. Let's compute CWM of 'num1' for 'com5' by hand as we learned above (i.e. removing species which has NA from the calculations)

```
com5 <- dummy$abun[5,]
com5[3] <- 0
relab.com5 <- com5 / sum(com5)
sum(relab.com5 * dummy$trait[, 1], na.rm = T)
```

```
## [1] 4.5
```

This is exactly the value obtained when running the object `ex2` above. So with quantitative traits the functcomp function indeed removes the species for which traits are missing, i.e. with NA. For binary and categorical traits this is not done. So, in summary, *when you have missing values in some binary of categorical traits the function functcomp* the function does different things. Luckily categorical traits are generally more easily available and there will be less NAs. But please do not forget this issue. If you are interested to follow the same approach, for both quantitative and categorical traits we suggest to solve by computing things by hand as shown in this section. If you have the traits organized as in the case of Carnivory, as a binary 0/1 value this will be easy. What to do if you have a factor with different levels, as 'fac1'?

The solution is to use the package `psych`, loaded at the beginning of this section, and do the following:

```
dummyfac1 <- dummy.code(dummy$trait$fac1)
rownames(dummyfac1) <- rownames(dummy$trait)
dummyfac1
```

```
##      C  A  B
## sp1  0  1  0
## sp2  0  1  0
```

```
## sp3  1  0  0
## sp4  0  0  1
## sp5  1  0  0
## sp6  1  0  0
## sp7  0  0  1
## sp8 NA NA NA
```

In the example we have created the so called 'dummy variables' (see Chapter 3 in the reference text book and R material Ch 3 out of a factorial trait (fac1). Based on this object `dummyfac1` you can now apply the same principle we used for Carnivory above. In this case, for simplicity, *we can remove the species 8 for both the traits and community data*, simply because of the missing value:

```
relab <- dummy$abun[,-8] / rowSums(dummy$abun[,-8]) #species 8 removed
fac1_A <- colSums(t(relab) * dummyfac1[-8, "A"])
fac1_B <- colSums(t(relab) * dummyfac1[-8, "B"])
fac1_C <- colSums(t(relab) * dummyfac1[-8, "C"])
cbind(fac1_A, fac1_B, fac1_C)
```

```
##          fac1_A     fac1_B     fac1_C
## com1  0.2500000 0.00000000 0.75000000
## com2  0.0000000 0.66666667 0.33333333
## com3  0.6666667 0.00000000 0.33333333
## com4  0.1250000 0.00000000 0.87500000
## com5  0.0000000 0.37500000 0.62500000
## com6  0.2000000 0.06666667 0.73333333
## com7  0.7272727 0.27272727 0.00000000
## com8  0.0000000 0.11111111 0.88888889
## com9  0.4545455 0.45454545 0.09090909
## com10 0.3636364 0.54545455 0.09090909
```

```
rowSums(cbind(fac1_A, fac1_B, fac1_C))
```

```
##  com1  com2  com3  com4  com5  com6  com7  com8  com9 com10
##     1     1     1     1     1     1     1     1     1     1
```

if you compare these results with the one obtained by using `functcomp` you will notice that *now the sum of the values for each community is, indeed, equal to 1.*

5.1.4 Calculation of CWM with real data

Let's now use real data and let's start with the NE Spain data described above. Let's first check the dimension of the objects and if the species names are the same in the plot composition data and in the trait data:

```
dim(spxp)
```

```
## [1] 134  60
```

```
dim(spxt)
```

```
## [1] 134  10
```

```
dim(envxp)
```

```
## [1] 60  3
```

```
rownames(spxp) == rownames(spxt)
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [76] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [106] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [121] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Let's also now have a look at the data. This is a glimpse at the 'species x plot matrix', for example the first 6 species in the 5th vegetation belt, i.e. the last 12 columns in the matrix. You will see that there are a lot of zero, as species from one vegetation belt are not in all vegetation belts.

```
spxp[1:6, 49:60]
```

```
##          5nograz.1 5nograz.2 5nograz.3 5nograz.4 5highgraz.1 5highgraz.2
## Acercamp         0         0         0         0           0           0
## Achimill         0         0         0         0          46          56
## Aegigeni         0         0         0         0           0           0
## Alchhybr         0         0         0         0          75           0
## Anemhepa        62         0        28         0           1           0
## Anthmont         0         0         0         0           0           0
##          5highgraz.3 5highgraz.4 5littlegraz.1 5littlegraz.2 5littlegraz.3
## Acercamp           0           0             0             0             0
## Achimill          47           2             3             2             2
## Aegigeni           0           0             0             0             0
## Alchhybr          10           0             0             0             0
## Anemhepa           0           0            73            38             0
## Anthmont           0           0             0             0             0
##          5littlegraz.4
## Acercamp             0
## Achimill             0
## Aegigeni             0
## Alchhybr             0
## Anemhepa             8
## Anthmont             0
```

Let's also now have a look at the trait data. Again, only the first 6 species for simplicity:

```
head(spxt)
```

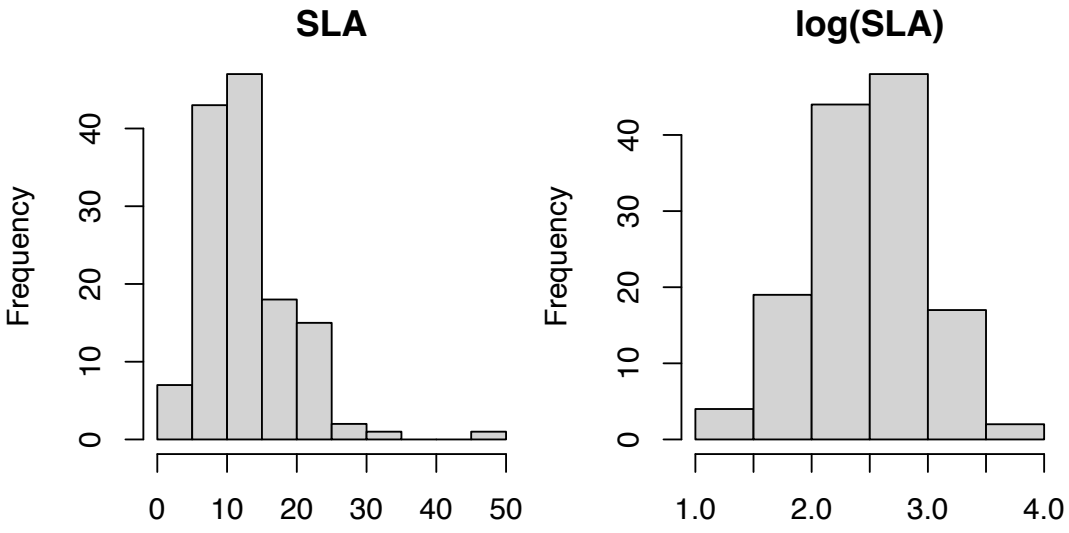
```
##          GrowhtForm LEG  SLA LF_Th LF_G LF_H LF_hCh LF_wCh LF_NP LF_P
## Acercamp      shrub   0 15.7     0    0  0.0    0.0      0     0    1
## Achimill       forb   0 14.6     0    0  0.5    0.5      0     0    0
## Aegigeni      grass   0 15.2     1    0  0.0    0.0      0     0    0
## Alchhybr       forb   0 19.0     0    0  1.0    0.0      0     0    0
## Anemhepa       forb   0 12.9     0    0  1.0    0.0      0     0    0
## Anthmont       forb   1 13.5     0    0  1.0    0.0      0     0    0
```

You can now see what type of trait information we have at hand. First we have one categorical trait *"GrowthForm"*, with 4 levels (grass, forb, i.e. herbs that are non grasses and subshrubs, small shrubs like thyme, shrub, including small trees). We then have data on whether the species are legumes *("LEG")*, which is a binary 0/1 trait, and the specific leaf area *("SLA")* which is quantitative. We then have a trait, which is Life Form *("LF_")*, which is introduced as a dummy variable, with fuzzy coding. This mean that the different life forms categories, generally following Raunkiaer scheme (https://en.wikipedia.org/wiki/Raunki%C3%A6r_plant_life-form), are represented by a different column (this includes Th=Therophytes, G=Geophytes, H=Hemicryptophytes, Ch=Chamaephytes, here divided in herbaceuous "h" and woody "w", NP=nano-Phanerophytes and P=Phanerophytes). Please see the link above for a detailed description of these groups. Each species can be part of different groups, althouh most of the times each species is only in one group. For example Acercamp, i.e. Acer campestris, i.e. a Phanerophytes, so it has the value 1 in this category. On the other hand some species, such as Achimill, Achillea millefolium, can be both an Hemicryptophytes and a Chamaephytes. In this case, we assign 0.5 in each column. Notice that the total value per species, for the LF trait has to sum up to 1 (see the Chapter 3 and 5 in the reference book for further explanations).

```
rowSums(spxt[, 4:10]) == 1 #sorry we do not show the results as they occupy a lot of space! but, y
```

We can also check if all quantitative traits are more or less normally distributed and if a log transformation would help improve normality

```
par(mfrow = c(1, 2))
par(mar = c(2, 4, 2, 0.5))
hist(spxt$SLA, main = "SLA", xlab = "")
hist(log(spxt$SLA), main = "log(SLA)", xlab = "")
```



Hence is better to log-transform the SLA data, for example in the following way (but be careful to *run the following line only once!*)

```
spxt$SLA <- log(spxt$SLA)
```

Alternatively you can create a new variable.

We can now briefly explore the 'envxp' data (the first 14 lines for example), containing the information, for each plot, to which vegetation belt *(vegbelt)* they belong (1=driest and warmest, 5=wettest and coldest) and the *grazing* pressure (0=no grazing, 2=high grazing and 1=intermediate grazing). For the analyses we provide also, for each vegetation belt, a

moisture index (the highest value meaning the highest ratio between rainfall and evapotranspiration):

```
envxp[1:14,]
```

```
##               vegbelt grazing moisture.index
## 1nograz.1           1       0           0.39
## 1nograz.2           1       0           0.39
## 1nograz.3           1       0           0.39
## 1nograz.4           1       0           0.39
## 1highgraz.1         1       2           0.39
## 1highgraz.2         1       2           0.39
## 1highgraz.3         1       2           0.39
## 1highgraz.4         1       2           0.39
## 1littlegraz.1       1       1           0.39
## 1littlegraz.2       1       1           0.39
## 1littlegraz.3       1       1           0.39
## 1littlegraz.4       1       1           0.39
## 2nograz.1           2       0           0.50
## 2nograz.2           2       0           0.50
```

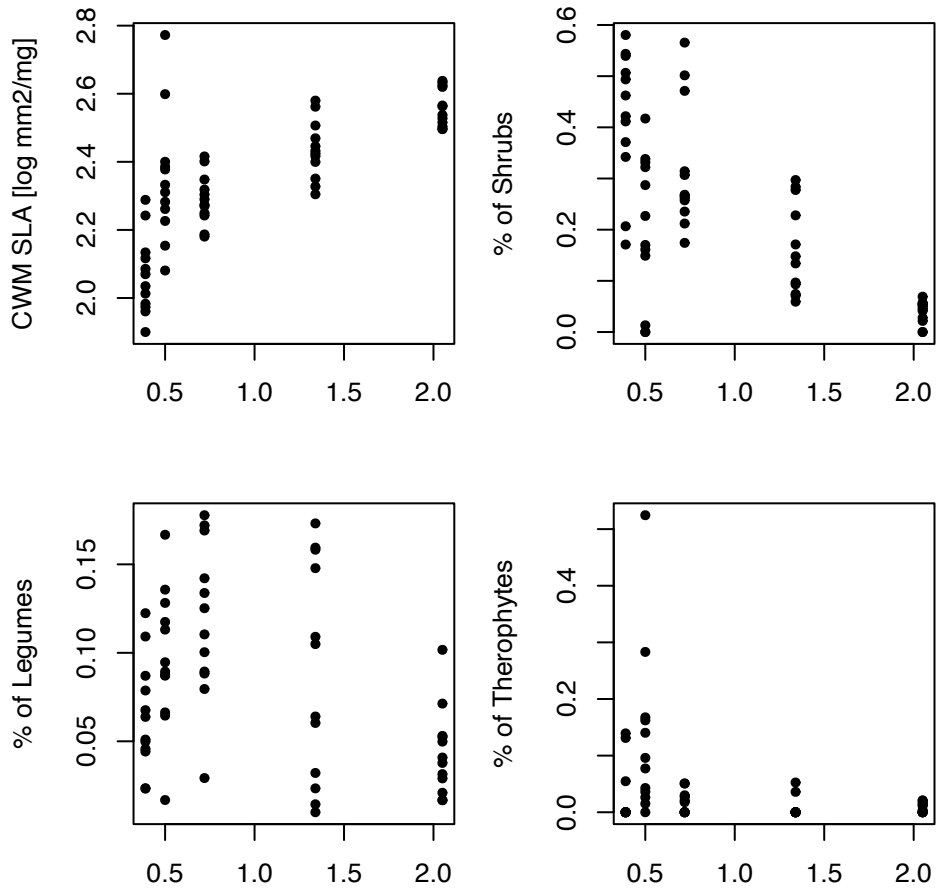
We are now *ready to compute CWM*. We have no NAs so our life will be very easy. *Notice* that, in this case we decided to log transform the abundance data, to decrease the importance of more dominant species (using *log(x+1)* on the species x community matrix). The results shown below are basically the same with and without such a transformation (you can try), but we show it below just to demonstrate how such transformation of data can be applied.

```
resCWM <- functcomp(spxt, log(t(spxp) + 1), CWM.type = "all")
# resCWM<-functcomp(spxt, t(spxp), CWM.type = "all")#option without log tranformation
head(resCWM)
```

```
##             GrowhtForm_forb GrowhtForm_grass GrowhtForm_shrub
## 1nograz.1         0.1744147       0.07487008        0.5804423
## 1nograz.2         0.2265858       0.08545117        0.5065337
## 1nograz.3         0.1416868       0.13598654        0.4622772
## 1nograz.4         0.1486632       0.11541070        0.4937316
## 1highgraz.1       0.2381015       0.06609128        0.4217385
## 1highgraz.2       0.2687062       0.07414532        0.4115459
##             GrowhtForm_subshrub     LEG_0      LEG_1      SLA LF_Th LF_G
## 1nograz.1             0.1702729 0.9557956 0.04420439 2.085576     0    0
## 1nograz.2             0.1814293 0.9501596 0.04984043 2.116583     0    0
## 1nograz.3             0.2600494 0.8776035 0.12239648 1.983106     0    0
## 1nograz.4             0.2421945 0.9324006 0.06759940 1.960632     0    0
## 1highgraz.1           0.2740688 0.9129762 0.08702385 2.069951     0    0
## 1highgraz.2           0.2456026 0.9764965 0.02350351 2.133959     0    0
##                  LF_H     LF_hCh    LF_wCh     LF_NP      LF_P
## 1nograz.1   0.1720212 0.07726351 0.1378888 0.3257382 0.2870883
## 1nograz.2   0.1753975 0.07808483 0.2068031 0.3045641 0.2351504
## 1nograz.3   0.1735466 0.06028072 0.3038955 0.3311344 0.1311428
## 1nograz.4   0.1542528 0.05361035 0.2984052 0.3464346 0.1472970
## 1highgraz.1 0.1321826 0.10444267 0.3055613 0.2614245 0.1963890
## 1highgraz.2 0.1590850 0.12445022 0.2752607 0.2000107 0.2411933
```

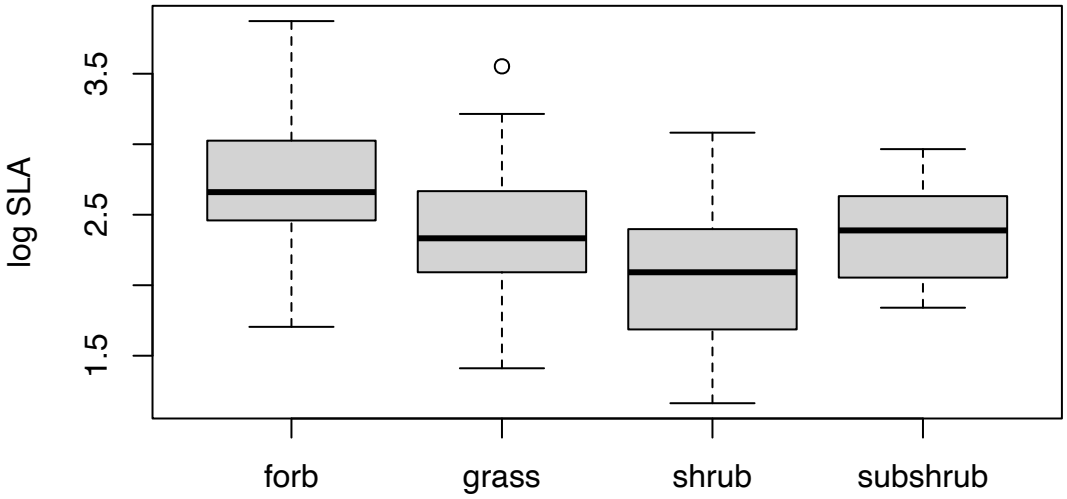
WELL DONE! we now have the CWM data. So….what shall we do with this information? Let's see if CWM changes along the climatic gradient. For example, let's explore visually a bit the results before running any analysis, just choosing some of the potential graphs (4 in this case):

```
par(mfrow = c(2, 2))
par(mar = c(3, 4, 2, 1))
plot(envxp$moisture.index, resCWM$SLA, xlab = "moisture index",
     ylab = "CWM SLA [log mm2/mg]",  pch = 20)
plot(envxp$moisture.index, resCWM$GrowhtForm_shrub,  xlab = "moisture index",
     ylab = "% of Shrubs",  pch = 20)
plot(envxp$moisture.index, resCWM$LEG_1, xlab = "moisture index",
     ylab = "% of Legumes", pch = 20)
plot(envxp$moisture.index, resCWM$LF_Th, xlab = "moisture index",
     ylab = "% of Therophytes", pch = 20)
```



We can already reach some conclusions while looking at these figures. The CWM SLA is increasing with moisture index, likely because of a decrease in shrub species with moisture. If fact, the different growth forms tend to have different SLA. You can test this by using this line:

```
boxplot(spxt$SLA ~ spxt$GrowhtForm, ylab = "log SLA", xlab = "")
```



We can possibly also expect that there is slightly more leguminous species at the intermediate moisture conditions and that the presence of annuals (i.e. therophytes) is slightly higher in the two driest belts [Note that in the second driest site, with moisture index=0.5, there was overall a slightly greater grazing history that in the other sites, hence the generally strange behaviour of the points in these plots, with respect to the general trends].

We can now have a specific look at the statistics and we can also include grazing in the analysis. For example we can use a simple linear model (although maybe a REML, restricted maximum likelihood model, would have been more accurate) where we test the effect of moisture and grazing intensity on the CWM for SLA:

```
summary(lm(resCWM$SLA ~ moisture.index * grazing, data = envxp))
```

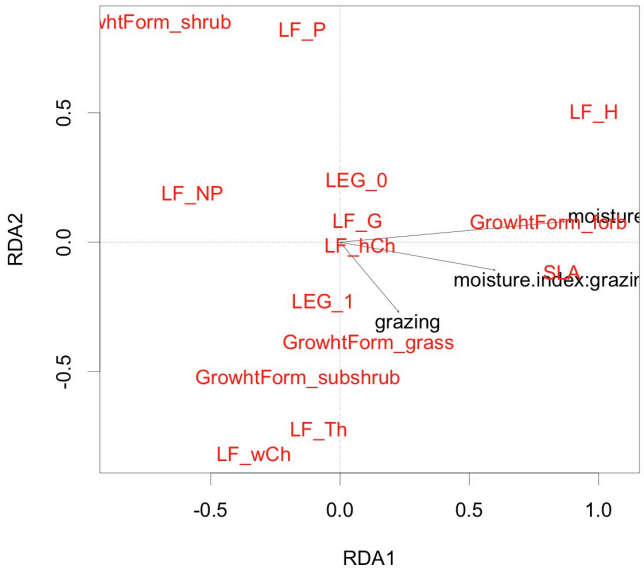
```
##
## Call:
## lm(formula = resCWM$SLA ~ moisture.index * grazing, data = envxp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30018 -0.07147  0.00094  0.06982  0.46678
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.99436    0.05060  39.413  < 2e-16 ***
## moisture.index            0.30168    0.04302   7.013 3.29e-09 ***
## grazing                   0.11652    0.03920   2.973  0.00435 **
## moisture.index:grazing -0.07248    0.03332  -2.175  0.03384 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1305 on 56 degrees of freedom
## Multiple R-squared:  0.589,  Adjusted R-squared:  0.567
## F-statistic: 26.75 on 3 and 56 DF,  p-value: 7.237e-11
```

This shows that in general the strongest effect (i.e. 'filter') on CWM for SLA is by the moisture index but grazing also modifies CWM for SLA, specifically increasing it (likely because of an increase in therophytes and decrease in woody species). However the effect of grazing on CWM for SLA depends on moisture (see the interaction moisture.index:grazing), with a weaker effect in more moist conditions.

As we shown in Chapter 4 of the reference book, the presence of multiple CWM values for each plot, allows us to use multivariate analyses for the representation of the data

and statistical analysis. For example we could use RDA (see Chapter 4 and Kleyer et al. (2012)).

```
library(vegan) # it is already loaded when you use the package 'FD'
rdaNEspain.all <- rda(resCWM ~ moisture.index * grazing, data = envxp)
plot(rdaNEspain.all, type = "n", scaling = "sites")
text(rdaNEspain.all, dis = "cn", scaling = "sites")
text(rdaNEspain.all, dis = "sp", scaling = "sites", col = "red")
```



In the figure, we can see which CWM values are associated with an increase of moisture index, mostly corresponding to the first axis of the RDA, and grazing partially on the second axis. For example, glowworm 'forbs' (GrowhtForm_forb) are expected to be favoured by higher moisture (i.e. high moisture index), while Phanerophytes (LF_P) are expected to increase with low grazing and in relatively drier conditions (i.e. slightly low moisture index).

A global statistical test would be done simply as in the following script. Please take into account that we prefer here not to dive too deeply into the functionality of multivariate analyses, for which we highly recommend the page done by David Zeleny, specifically for R tool: https://www.davidzeleny.net/anadat-r/doku.php/en:start. However, we will first run these analyses and provide some basic understanding of the approach:

```
rdaNEspain0 <- rda(resCWM ~ 1, data = envxp)
rdaNEspain.all <- rda(resCWM ~ moisture.index * grazing, data = envxp)
ordistep(rdaNEspain0, scope = formula (rdaNEspain.all), direction = 'forward')
```

```
##
## Start: resCWM ~ 1
##
##                   Df      AIC       F Pr(>F)
## + moisture.index   1 -133.851 56.0928  0.005 **
## + grazing          1  -98.456  5.2501  0.010 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step: resCWM ~ moisture.index
##
```

```
##           Df     AIC      F Pr(>F)
## + grazing  1 -142.55 11.123  0.005 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step: resCWM ~ moisture.index + grazing
##
##                           Df     AIC      F Pr(>F)
## + moisture.index:grazing  1 -144.33 3.6451  0.005 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step: resCWM ~ moisture.index + grazing + moisture.index:grazing

## Call: rda(formula = resCWM ~ moisture.index + grazing +
## moisture.index:grazing, data = envxp)
##
##               Inertia Proportion Rank
## Total          0.2011     1.0000
## Constrained    0.1208     0.6006    3
## Unconstrained  0.0803     0.3994   11
## Inertia is variance
##
## Eigenvalues for constrained axes:
##    RDA1    RDA2    RDA3
## 0.10552 0.01155 0.00369
##
## Eigenvalues for unconstrained axes:
##     PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10
## 0.03191 0.02523 0.00746 0.00610 0.00366 0.00251 0.00193 0.00072 0.00060 0.00013
##    PC11
## 0.00004
```

```
RsquareAdj (rdaNEspain.all)$adj.r.squared
```

```
## [1] 0.5792475
```

First notice that we used the function `ordistep`, which is basically a stepwise regression analyses applied to multivariate analyses, which resembles the function `step` for linear models. In this case we selected a 'forward' selection, see argument 'forward' above. The approach is working by a comparing a model with all predictors ('rdaNEspain.all') with an 'empty' model ('rdaNEspain0', with no predictors). The results indicate that both grazing and the moisture index are kept into the model, including their interactions (moisture.index:grazing), for being significant. So the final model is indicated in `Call: rda(formula = resCWM ~ moisture.index + grazing + moisture.index:grazing, data = envxp)`. The analysis further provides a lot of interesting information about constrained and unconstrained axes, their eigenvalues etc. While important we prefer here not to explain their meaning and do refer to the interesting page by David Zeleny indicated above.

At the same time we can see also how to understand the amount of variability explained by this model, simply using the `RsquareAdj` function. The resulting model keeps moisture and grazing, and their interaction in the model, for a R2~0.6, similarly to the linear model we considered above for SLA in a simple linear model.

5.1.5 Can we trust the p-value of CWM-based analyses?

In the reference text book, in Chapter 5, we provide an extensive discussion on some potential problems using analyses with CWM. Some authors have suggested that CWM should not be

used because the analyses using CWM provide too optimistic p-values. As summarized by David Zeleny and in the reference text book (Zelený, 2018), we think that the question is *not* if we can trust the the p-value of CWM-based analyses but rather which type of questions are asked. If we are interested in observing changes in CWM along gradient, whatever are the causes, then analyses on CWM are robust and interesting. If the question is, for example, if the MAJORITY of species in drier habitats have given traits, then a different type of analyses should be done, basically the analyses at the species level presented in Chapter 3 and in the R material Ch 2. Alternatively, for this type of questions we can have two other solutions still using the CWM. One simple solution is presented below. For other approaches see David's page: https://www.davidzeleny.net/anadat-r/doku.php/en:traits

Let's now focus on the suggestion by David Zeleny (2018) on how to run a conservative tests to evaluate the significance of environmental factors on CWM values. The suggestion by Zeleny can be summarized as to run randomizations tests and, basically, verify if the R^2 of a given model (for example those obtained in the previous section) is higher than expected by chance. Randomizations can be done in different ways, but we suggest that a simple and effective one is by simply shuffling species names in the trait matrix (we tested the power of this test and it provided the best results compared to other approaches, not shown here). For simplicity here we present only the test of the effect of moisture on CWM for SLA.

Randomizations in R can be done in different ways and we will use a relative simple and common one, 'loops'. We assume the users are, more or less, familiar with loops in R. Here we make a loop, with 999 randomizations. For each of these randomizations (i=1, i=2, i=3….) we randomize species names in the species trait matrix and make a new regression between a CWM and the environmental variable, using the CWM obtained with such randomization. Here we test the relationship between CWM for SLA and the moisture index. The results of each of these 999 models are stored in the object `EXPECTED.R2.SLA.moist`.

```
resCWM <- functcomp(spxt, log(t(spxp) + 1), CWM.type = "all") #this are the observed results
OBS.R2.SLA.moist <- summary(lm(resCWM$SLA ~ envxp$moisture.index))$r.squared #observed R2
EXPECTED.R2.SLA.moist <- vector()
for(i in 1:999) {
  random.names <- sample(rownames(spxt))
  spxt.rand1 <- spxt
  rownames(spxt.rand1) <- random.names
  spxt.rand.final.i <- spxt.rand1[sort(rownames(spxt.rand1)),]
  EXP.resCWM <- functcomp(spxt.rand.final.i, log(t(spxp) + 1), CWM.type = "all")
  EXPECTED.R2.SLA.moist[i]<-summary(lm(EXP.resCWM$SLA~envxp$moisture.index))$r.squared
}
```

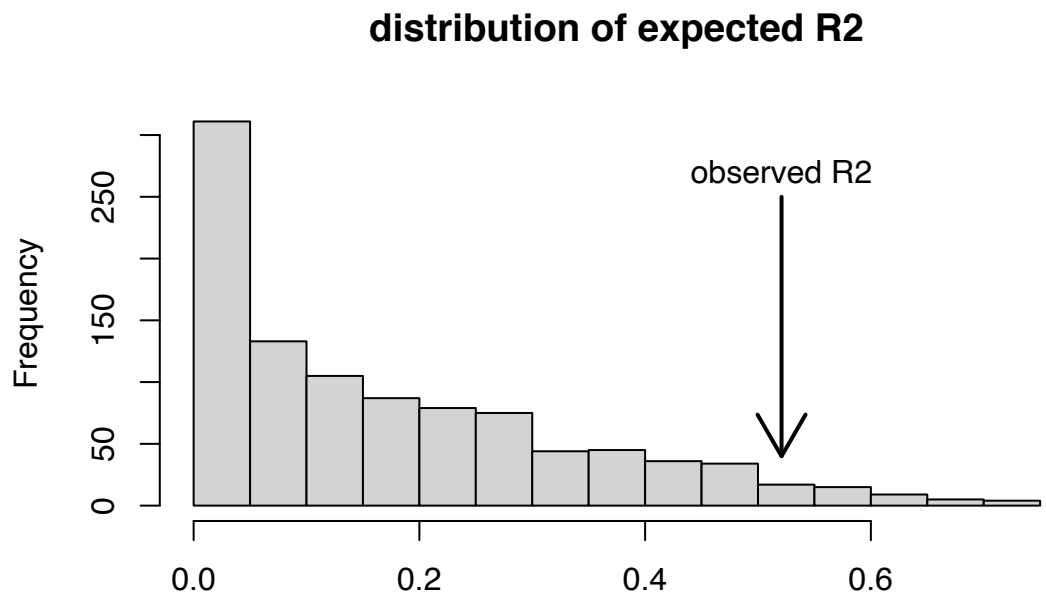
This takes definitely a while (couple of minutes) to run, in case you want to run the script on your own. Let's first have a look on how the randomization in species names results in a change in trait values for each species.

```
head(spxt.rand.final.i) #this corresponds, for example, to the very last randomization, i.e the 99
```

```
##           GrowhtForm LEG      SLA LF_Th LF_G LF_H LF_hCh LF_wCh LF_NP LF_P
## Acercamp       shrub   0 1.386294     0    0    0      0      0     0    1
## Achimill    subshrub   0 2.965273     0    0    0      1      0     0    0
## Aegigeni       grass   0 1.410987     0    0    1      0      0     0    0
## Alchhybr        forb   0 2.517696     0    0    1      0      0     0    0
## Anemhepa       shrub   0 2.370244     0    0    0      0      0     0    1
## Anthmont        forb   0 2.397895     0    0    1      0      0     0    0
```

We can see that the species are in the same order as before (which we need for the calculations), but the trait values are different! so we did what we wanted! We can now see how frequently the observed values of the R^2 is bigger than the expected values. First graphically and then in numbers

```
hist(EXPECTED.R2.SLA.moist, main = "distribution of expected R2", xlab = "")
arrows(OBS.R2.SLA.moist, 250, OBS.R2.SLA.moist, 40, lwd = 2)
text(OBS.R2.SLA.moist, 270, "observed R2")
```




```
sum(OBS.R2.SLA.moist > EXPECTED.R2.SLA.moist) / 1000
```

```
## [1] 0.955
```

This last value represents the proportion of cases in which the observed R2 was higher than expected. In our case, after running the loop several times we generally found that the proportion was around 0.95, i.e. 95% of the cases. With this we can finally compute a p-value providing the significance of the relationship between CWM-SLA and the moisture index computed, after randomizations, as:

```
pval <- 1 - sum(OBS.R2.SLA.moist > EXPECTED.R2.SLA.moist) / 1000
pval
```

```
## [1] 0.045
```

In practice it tells us that the relationship between CWM-SLA and the moisture index is significant, or very close to significant (depending on the specific randomizations run, try the script few times to verify it, in case you are interested). Please take into account that this approach using randomizations of species names can be applied to different type of analyses, included the RDA described above, or any type of other statistical tool.

Most important, as you can see the p-value just obtained pval is not as good as in simple regression model, which we already used above, and remind here:

```
summary(lm(resCWM$SLA ~ moisture.index, data = envxp))
```

```
##
## Call:
## lm(formula = resCWM$SLA ~ moisture.index, data = envxp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -0.30018 -0.08382 -0.00155  0.05706  0.54705
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.11088    0.03395   62.17  < 2e-16 ***
## moisture.index  0.22919    0.02886    7.94 7.71e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1385 on 58 degrees of freedom
## Multiple R-squared:  0.5209, Adjusted R-squared:  0.5126
## F-statistic: 63.05 on 1 and 58 DF,  p-value: 7.708e-11
```

As mentioned above, and more in depth in the reference text book (Chapter 5), we to interpret these differences between these two approaches very carefully, as the two test are considering different null hypotheses (see Zelený, 2018). The model *without* randomizations, i.e. the last one displayed, test the null-hypothesis that there is some change in CWM along the studied gradient. And there is definitely a marked change which cannot be denied. The model *with* randomization tests the null-hypothesis that the change in CWM along the gradient considered depends on the majority of species, i.e. not just on the replacement of few dominant species. This test resembles much more the test done at the *species level* in Chapter 4 of the reference text book and R material Ch 3. Both tests are interesting but the interpretation IS different.

5.2 Functional Diversity (FD)

In this exercise we will learn how to compute different indices of Functional diversity (*FD*). This exercise follows the second part of Chapter 5 of the reference textbook, so all theoretical and mathematical issues beyond the indices described in this exercise can be found there. We will work with invented data as used in the Chapter 5 of the reference textbook (data created below) and also field data from an climatic gradient in NE Spain, which was already described in detail in the CWM exercise (R material 5.1: CWM). In Chapter 3, and the relative exercise, we already explained how to compute trait dissimilarity between species pairs, and thus in this exercise we assume that users have already an idea how to compute and interpret such trait dissimilarity. This file will first cover the use of the `dbFD` function, with invented data and then with the NE Spain data. Then we will first introduce other indices that can be computed with the package *picante* (FD from Petchey and Gaston (2002), which is the same of Faith index, and MPD and MNTD). Then, we will show that MPD computed with picante presents some unexpected results, in connection with the Rao index, and we will provide some solutions to overcome it. Here we will also see that the Rao index computed by the dbFD function is expressing a particular form of the Rao index, i.e. trait variance. Finally, we will learn how to compute alpha, beta and gamma functional diversity with the Rao index.

5.2.1 Data

See explanations about the data from NE Spain introduced in the R material 5, on Community Weighted Mean (CWM) (5.1). Let's open the data if they are not already opened:

```
spxp <- read.table(here::here("data", "chapter5", "speciesXplotsNE.txt"), row.names = 1, header =
spxt <- read.table(here::here("data", "chapter5", "speciesXtraitsNE.txt"), row.names = 1, header =
envxp <- read.table(here::here("data", "chapter5", "environXplotsNE.txt"), row.names = 1, header =
```

Notice that for all files we specifically asked that the names of species and plots are not part of the matrix, but they are used (in this case) as names of the rows (`row.names=1`) and as

column names (`header=T`). Be careful to use these options carefully when you consider your data, because they could be in a different arrangement.

5.2.2 Required packages and additional functions

We also need to load various packages in order to run the analyses. Please ensure that you have installed these packages in your R library before you load them.

```
library(FD)
library(picante)
```

```
## Warning: package 'picante' was built under R version 4.0.2
```

```
## Warning: package 'nlme' was built under R version 4.0.2
```

We will also need two ad-hoc functions, 'melodic' and 'Rao'

```
source(here::here("data", "chapter5", 'melodic.r')) #which requires the package 'nlme'
```

```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
source(here::here("data", "chapter5", 'Rao.r'))
```

5.2.3 Calculation of FD with the dbFD function

As well summarized in the help function of `?dbFD`, dbFD implements a flexible distance-based framework to compute multidimensional functional diversity (FD) indices. dbFD returns the three FD indices of Villéger et al. (Villéger et al., 2008): *functional richness (FRic)*, *functional evenness (FEve)*, and *functional divergence (FDiv)*, as well as *functional dispersion (FDis; Laliberté and Legendre, 2010)*, *Rao's quadratic entropy (Q)* (Botta-Dukát, 2005), a functional group richness (FGR) (number of functional groups, see Chapter 3 and Reference material Ch3 and Petchey and Gaston (2006)), and the community-level weighted means of trait values (CWM; e.g. Lavorel et al., 2007). Some of these FD indices consider species abundances (see Fig. 5.4 in the reference book).

As we just saw, the function `dbFD` is very practical because it computes *a lot of indices of FD at once*. Most of these indices are introduced and discussed in the reference textbook,

in Chapter 5. We will thus learn how to apply the function `dbFD`, using first some of the examples available in the help of the function, i.e. from `?dbFD`. As in the exercises with CWM, the examples for the function consider invented data, the 'dummy' data, containing both a 'species x community' matrix and a 'species x trait' matrix.

```
library(FD)
dummy$abun # Species x community matrix
```

```
##       sp1 sp2 sp3 sp4 sp5 sp6 sp7 sp8
## com1    1   1   0   0   4   2   0   0
## com2    0   0   0   2   1   0   0   5
## com3    2   0   0   0   0   1   0   3
## com4    1   0   7   0   0   0   0   0
## com5    0   0   2   3   3   0   0   0
## com6    0   3   0   0   5   6   1   6
## com7    3   5   0   3   0   0   0   0
## com8    0   0   0   0   6   2   1   2
## com9    4   1   1   3   0   0   2   0
## com10   0   4   1   0   0   0   6   1
```

```
dummy$trait # Species x trait matrix
```

```
##     num1 num2 fac1 fac2 ord1 ord2 bin1 bin2
## sp1  9.0  4.5    A    X    3    2    0    1
## sp2  8.1  6.0    A    Z <NA>    1    0    1
## sp3   NA  2.3    C    Y    5    3    1    1
## sp4  3.2  5.4    B    Z    1    7    0    0
## sp5  5.8  1.2    C    X    2    6   NA    0
## sp6  3.4  8.5    C    Y    2    1    1    1
## sp7  7.5  2.1    B    X    3    2    1    0
## sp8  4.3  6.5 <NA>    Z    1    3    0    1
```

Notice that the function `dbFD`, while extremely useful, has also many strict requirements. The species x community matrix should be a data frame and the order and name of species should be *exactly* the same in the 'species x community matrix' and in the Species x trait matrix.

With this type of data we can use the function in the same way as the function `functcomp` already discussed in the CWM section (R material Ch 4 As we discussed in that exercise it is very important to check that the species that you have in these 'species x community' matrix and 'species x trait' matrices should be the same. In this case we know that they are the same, so we can already run the code:

```
ex1 <- dbFD(dummy$trait, dummy$abun)
```

```
## Species x species distance matrix was not Euclidean. 'sqrt' correction was applied.
## FEVe: Could not be calculated for communities with <3 functionally singular species.
## FRic: To respect s > t, FRic could not be calculated for communities with <3 functionally singu
## FRic: Dimensionality reduction was required. The last 5 PCoA axes (out of 7 in total) were remo
## FRic: Quality of the reduced-space representation (based on corrected distance matrix) = 0.6108
## FDiv: Could not be calculated for communities with <3 functionally singular species.
```

First we see that the function is very "talkative", i.e. it explains a lot of things while running. We can eventually *switch-off the messages* by using `messages=F`, as we will do in the next examples. In the meantime we can see that FEve (functional evenness) cannot be computed in communities with less than 3 species. Similarly as FRic (functional richness) and FDiv

(functional divergence) cannot be computed if there is less that 3 unique species, meaning that they should at least have one different trait value. Other important messages include the number of PCoA axes considered. Out of the 7 considered (i.e. with 8 species, the number of PCoA is 7) a total of 2 axes are retained to reflect trait differences between species (see Chapter 3 in the reference book and R material Ch 3. In other words the PCoA has synthesized all traits into two multivariate axes.

Let's now see the results. The results are stored in the object ex1, which is a *'list'*. As such if you need a particular object you can extract it, for example as `ex1$FRic`:

```
class(ex1)
```

```
## [1] "list"
```

```
ex1
```

```
## $nbsp
##  com1  com2  com3  com4  com5  com6  com7  com8  com9 com10
##     4     3     3     2     3     5     3     4     5     4
##
## $sing.sp
##  com1  com2  com3  com4  com5  com6  com7  com8  com9 com10
##     4     3     3     2     3     5     3     4     5     4
##
## $FRic
##        com1        com2        com3        com4        com5        com6
## 0.174201349 0.102097174 0.002157642          NA 0.143151204 0.231083703
##        com7        com8        com9       com10
## 0.073683375 0.174220613 0.337446205 0.228904118
##
## $qual.FRic
## [1] 0.6108352
##
## $FEve
##      com1      com2      com3      com4      com5      com6      com7      com8
## 0.8432334 0.4628635 0.8659657        NA 0.9081209 0.8651734 0.7681991 0.7606650
##      com9     com10
## 0.7994638 0.4944601
##
## $FDiv
##      com1      com2      com3      com4      com5      com6      com7      com8
## 0.8429221 0.8593250 0.6303031        NA 0.7631346 0.8966687 0.8356095 0.8681163
##      com9     com10
## 0.7015118 0.9712554
##
## $FDis
##      com1      com2      com3      com4      com5      com6      com7      com8
## 0.3481687 0.1670560 0.2375808 0.1146261 0.3211366 0.3302330 0.2532751 0.2877931
##      com9     com10
## 0.3421687 0.3503927
##
## $RaoQ
##       com1       com2       com3       com4       com5       com6       com7
## 0.12835440 0.04063622 0.06497224 0.03003235 0.11858388 0.11738479 0.07868047
##       com8       com9      com10
## 0.09308505 0.12431265 0.12490783
##
## $CWM
```

```
##           num1     num2 fac1 fac2 ord1 ord2 bin1 bin2
## com1  5.887500 4.037500    C    X    2    6    0    0
## com2  4.212500 5.562500    B    Z    1    3    0    1
## com3  5.716667 6.166667    A    Z    1    3    0    1
## com4  9.000000 2.575000    C    Y    5    3    1    1
## com5  4.500000 3.050000    C    X    2    7    0    0
## com6  5.095238 5.528571    C    Z    2    1    0    1
## com7  7.009091 5.427273    A    Z    3    1    0    1
## com8  5.245455 3.572727    C    X    2    6    1    0
## com9  6.870000 4.245455    A    X    3    2    0    1
## com10 7.427273 3.783333    B    X    3    2    1    1
```

```
ex1$FRic
```

```
##        com1        com2        com3        com4        com5        com6
## 0.174201349 0.102097174 0.002157642          NA 0.143151204 0.231083703
##        com7        com8        com9       com10
## 0.073683375 0.174220613 0.337446205 0.228904118
```

The function `dbFD` includes also the function `functcomp`, which we already discussed above. As such, for categorical and binary traits you might want to include the argument `CWM.type="all"` when running it, as we discussed also in the example where we calculated CWM in the R material Ch 5.1.

```
ex1 <- dbFD(dummy$trait, dummy$abun, CWM.type = "all", message = F)
# notice we add 'message=F' to avoid too many annoying messages from the function :)
ex1$CWM
```

```
##           num1     num2    fac1_A     fac1_B     fac1_C    fac2_X     fac2_Y
## com1  5.887500 4.037500 0.2500000 0.00000000 0.75000000 0.6250000 0.25000000
## com2  4.212500 5.562500 0.0000000 0.25000000 0.12500000 0.1250000 0.00000000
## com3  5.716667 6.166667 0.3333333 0.00000000 0.16666667 0.3333333 0.16666667
## com4  9.000000 2.575000 0.1250000 0.00000000 0.87500000 0.1250000 0.87500000
## com5  4.500000 3.050000 0.0000000 0.37500000 0.62500000 0.3750000 0.25000000
## com6  5.095238 5.528571 0.1428571 0.04761905 0.52380952 0.2857143 0.28571429
## com7  7.009091 5.427273 0.7272727 0.27272727 0.00000000 0.2727273 0.00000000
## com8  5.245455 3.572727 0.0000000 0.09090909 0.72727273 0.6363636 0.18181818
## com9  6.870000 4.245455 0.4545455 0.45454545 0.09090909 0.5454545 0.09090909
## com10 7.427273 3.783333 0.3333333 0.50000000 0.08333333 0.5000000 0.08333333
##          fac2_Z     ord1_1    ord1_2     ord1_3     ord1_5     ord2_1
## com1  0.1250000 0.00000000 0.7500000 0.12500000 0.00000000 0.37500000
## com2  0.8750000 0.87500000 0.1250000 0.00000000 0.00000000 0.00000000
## com3  0.5000000 0.50000000 0.1666667 0.33333333 0.00000000 0.16666667
## com4  0.0000000 0.00000000 0.0000000 0.12500000 0.87500000 0.00000000
## com5  0.3750000 0.37500000 0.3750000 0.00000000 0.25000000 0.00000000
## com6  0.4285714 0.28571429 0.5238095 0.04761905 0.00000000 0.42857143
## com7  0.7272727 0.27272727 0.0000000 0.27272727 0.00000000 0.45454545
## com8  0.1818182 0.18181818 0.7272727 0.09090909 0.00000000 0.18181818
## com9  0.3636364 0.27272727 0.0000000 0.54545455 0.09090909 0.09090909
## com10 0.4166667 0.08333333 0.0000000 0.50000000 0.08333333 0.33333333
##           ord2_2     ord2_3    ord2_6    ord2_7    bin1_0    bin1_1    bin2_0
## com1  0.12500000 0.00000000 0.5000000 0.0000000 0.2500000 0.2500000 0.5000000
## com2  0.00000000 0.62500000 0.1250000 0.2500000 0.8750000 0.0000000 0.3750000
## com3  0.33333333 0.50000000 0.0000000 0.0000000 0.8333333 0.1666667 0.0000000
## com4  0.12500000 0.87500000 0.0000000 0.0000000 0.1250000 0.8750000 0.0000000
## com5  0.00000000 0.25000000 0.3750000 0.3750000 0.3750000 0.2500000 0.7500000
## com6  0.04761905 0.28571429 0.2380952 0.0000000 0.4285714 0.3333333 0.2857143
```

```
## com7  0.27272727 0.00000000 0.0000000 0.2727273 1.0000000 0.0000000 0.2727273
## com8  0.09090909 0.18181818 0.5454545 0.0000000 0.1818182 0.2727273 0.6363636
## com9  0.54545455 0.09090909 0.0000000 0.2727273 0.7272727 0.2727273 0.4545455
## com10 0.50000000 0.16666667 0.0000000 0.0000000 0.4166667 0.5833333 0.5000000
##          bin2_1
## com1  0.5000000
## com2  0.6250000
## com3  1.0000000
## com4  1.0000000
## com5  0.2500000
## com6  0.7142857
## com7  0.7272727
## com8  0.3636364
## com9  0.5454545
## com10 0.5000000
```

In the second example of the help page for dbFD, it is possible to see a the argument `w`, which was discussed for the `gowdis` in the R material Ch3 and in Chapter 3 of the reference textbook. As a matter of fact the function `gowdis` is also included in the `dbFD` function and, although you do not see it, it is applied exactly as already introduced in the R material 3. This is the first step for the calculation of functional diversity.

The argument `w` allows us to give different weights to the traits, for example when you want some traits to be more important in the calculation of the dissimilarity between species and thus functional diversity (we discuss the importance of this weight in the exercises related to Chapter 3). The example 2 in the help function looks like this:

```
# add variable weights
w <- c(1, 5, 3, 2, 5, 2, 6, 1)
ex2 <- dbFD(dummy$trait, dummy$abun, w, corr = "cailliez", message = F)
# 'cailliez' correction is used because 'sqrt' does not work
```

In practice the second trait got 5 times more weight than the first one (the vector `w`, includes one value for each trait, reflecting the intended weight in the calculation; again see R material Ch 2.

It might happen that when running the dbFD function you get some sort of *errors*. For example if you run a line such as `ex2 <- dbFD(dummy$trait, dummy$abun, w, message=F)`, you will get one saying that the distance between species did not have Euclidean properties after the square root correction, which is automatically applied in the function. The PCoA analyses work better when the dissimilarity matrix (differences between species in terms of traits; R material Ch 2) has Euclidean properties and the dbFD function will not work unless this condition is met. In the case that the function does not work directly with your data, you can try other *corrections with the argument* `corr`. The help function provides enough information on this issue. In practice the effect of such correction is not very strong, at least in our experience, especially when there are enough species.

It is very important to notice that the *function dbFD works not only by providing a 'species x trait' matrix* like the `dummy$trait` in the example above. You can also provide directly a distance matrix calculated before hand, for example using the *Gower distance* (see Chapter 3; R material Ch 3). If you run the following example (ex3) you will thus get the same results as in the object ex1.

```
trait.d <- gowdis(dummy$trait) # Gower distance
ex3 <- dbFD(trait.d, dummy$abun, message = F)
ex1$FRic == ex3$FRic
```

```
##  com1  com2  com3  com4  com5  com6  com7  com8  com9 com10
##  TRUE  TRUE  TRUE    NA  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Similarly if you apply w in the calculation of the Gower distance:

```
trait.dw <- gowdis(dummy$trait, w) # Gower distance with a different weight for the traits
ex3w <- dbFD(trait.dw, dummy$abun, corr = "cailliez", message = F)
ex2$FRic == ex3w$FRic
```

```
##  com1  com2  com3  com4  com5  com6  com7  com8  com9 com10
##  TRUE  TRUE  TRUE    NA  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Although we already discussed what is a dissimilarity matrix in Chapter 3, and its corresponding R material Ch 3, in the next lines, we show once again how it does look:

```
trait.d # Gower distance
```

```
##           sp1       sp2       sp3       sp4       sp5       sp6       sp7
## sp2 0.2181884
## sp3 0.5240052 0.6678082
## sp4 0.6737443 0.5610028 0.8225701
## sp5 0.5291113 0.8145699 0.4862253 0.4843264
## sp6 0.6100161 0.5932587 0.2784736 0.7073925 0.6067323
## sp7 0.4484235 0.6863374 0.4848663 0.5575126 0.3023416 0.6187844
## sp8 0.4072834 0.2039443 0.5958904 0.2390962 0.5585525 0.4470207 0.7030186
```

In practice, this object shows the *functional distance, i.e. dissimilarity between each pair of species*, in this case expressed as an average dissimilarity over all (standardized) traits.

The function `dbFD` applies the Gower distance automatically if you use a species x trait matrix (such as 'spxt'). It will also work when proving a dissimilarity matrix, *unless such dissimilarity does not contain any NAs* (NAs in the dissimilarity matrix are not accepted!). This means that you can either provide a specific dissimilarity matrix after checking that there are no NAs or you can make sure that, in the 'species x trait' matrix, you have 'enough' trait values. In other words, this means that species pairs should have, at least, information for one common trait.

A special case occurs when only one trait is used, as a vector, and there is some NA. In this case the function works (and it will remove the species with missing values):

```
num1 <- dummy$trait[, 1] #take only one trait, as a vector, with missing values
names(num1) <- rownames(dummy$trait) #give "back" the species names
ex4 <- dbFD(num1, dummy$abun) #it works
```

```
## Warning: Species with missing trait values have been excluded.
## FEVe: Could not be calculated for communities with <3 functionally singular species.
## FDis: Equals 0 in communities with only one functionally singular species.
## FRic: Only one continuous trait or dimension in 'x'. FRic was measured as the range, NOT as the
## FDiv: Cannot not be computed when 'x' contains one single continuous trait or dimension.
```

In all cases, you need to provide the species names in the trait data. If you have only one trait, WITHOUT NAs, you can do

```
round(gowdis(dummy$trait["num2"]), 3)
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6   sp7
## sp2 0.205
## sp3 0.301 0.507
## sp4 0.123 0.082 0.425
```

```
## sp5 0.452 0.658 0.151 0.575
## sp6 0.548 0.342 0.849 0.425 1.000
## sp7 0.329 0.534 0.027 0.452 0.123 0.877
## sp8 0.274 0.068 0.575 0.151 0.726 0.274 0.603
```

```
#does not remove species names and keep the trait data as a matrix; #here we keep only 3 dec
ex.1trait.noNA <- dbFD(gowdis(dummy$trait[, "num2", drop = F]), dummy$abun,
                       message = F) #it works! no errors provided.
```

5.2.4 dbFD function with NE Spain data

Let's now apply the `dbFD` function to the NE Spain data already described in section 5.1 above. The data was already loaded already above. As in the exercise on CWM, we need to improve the normality of the trait SLA in the matrix.

```
spxt$SLA <- log(spxt$SLA) #improve the normality of the trait values
head(spxt)
```

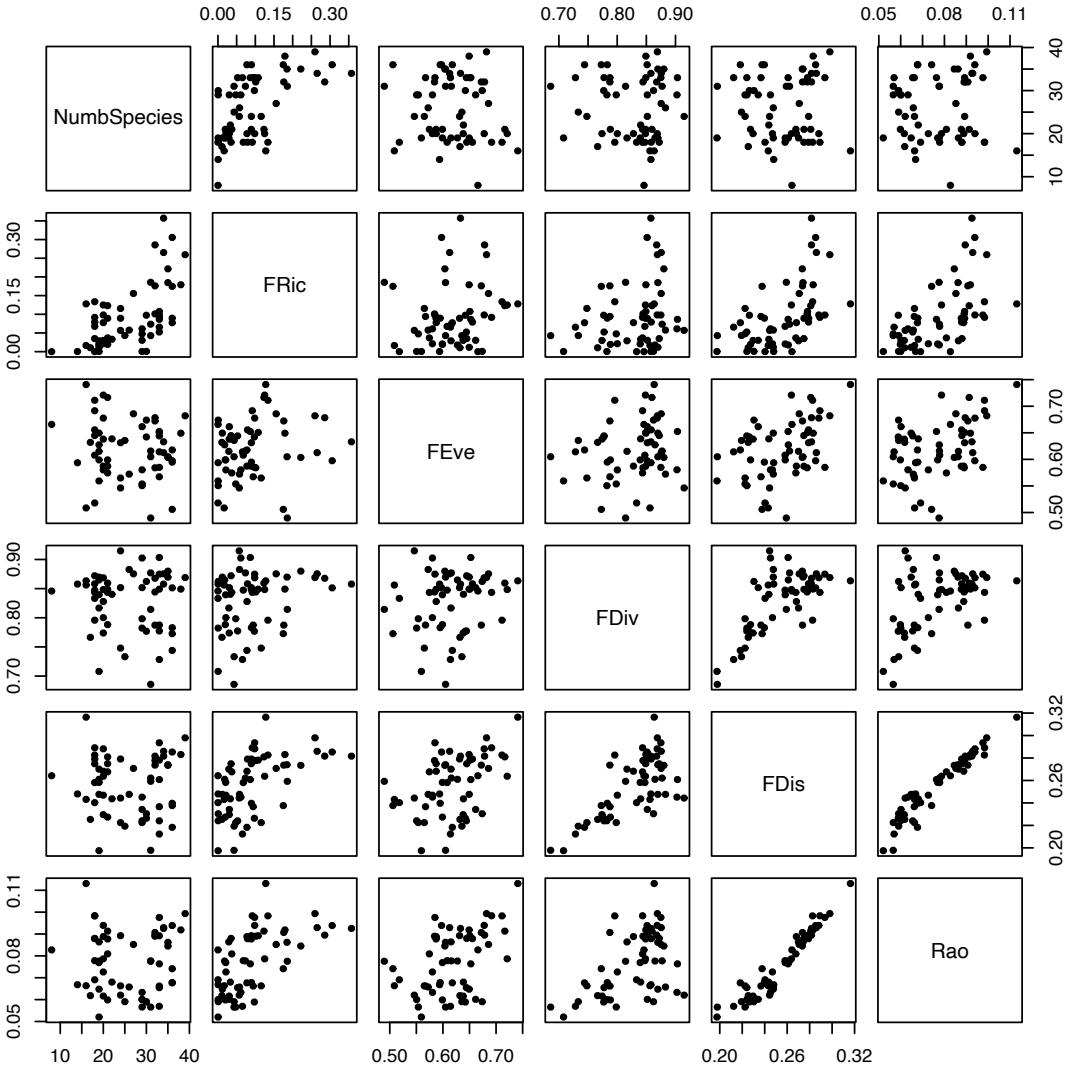
```
##          GrowhtForm LEG      SLA LF_Th LF_G LF_H LF_hCh LF_wCh LF_NP LF_P
## Acercamp      shrub   0 2.753661     0    0  0.0    0.0      0     0    1
## Achimill       forb   0 2.681022     0    0  0.5    0.5      0     0    0
## Aegigeni      grass   0 2.721295     1    0  0.0    0.0      0     0    0
## Alchhybr       forb   0 2.944439     0    0  1.0    0.0      0     0    0
## Anemhepa       forb   0 2.557227     0    0  1.0    0.0      0     0    0
## Anthmont       forb   1 2.602690     0    0  1.0    0.0      0     0    0
```

Let's directly use the `dbFD` function. We need first to recall that we have a fuzzy coding data in the trait matrix (spxt$LF_xx) and for now, for simplicity, let's forget about this type of trait data. *We can use thus only the first 3 columns of the trait matrix* for the time being (but see below for using all traits). Now that we know how the `dbFD` function works, i.e. it is used very similarly to the function `functcomp`, it is easy to have a lot of indices of FD computed in only one line! But for the time being we avoid computing again the CWM values, with the argument `calc.CWM = FALSE` and we also ask that the FRic will be standardized between 0 and 1, i.e. `stand.FRic = TRUE`.

```
resFD <- dbFD(spxt[, 1:3], log(t(spxp) + 1), message = F, calc.CWM = FALSE,
              stand.FRic = TRUE)
```

We can now explore a bit the results. First let's see how much *the different FD indices are correlated between them*:

```
important.indices <- cbind(resFD$nbsp, resFD$FRic, resFD$FEve, resFD$FDiv,
                           resFD$FDis, resFD$RaoQ)
colnames(important.indices) <- c("NumbSpecies", "FRic", "FEve", "FDiv", "FDis", "Rao")
pairs(important.indices, pch = 20)
```



We generally see that FRic is positively correlated to the number of species, as expected. Of course FDis and Rao give very similar results (they are actually the same index, with only a squaring difference, see Pavoine & Bonsall (2011)). Slight deviations from the perfect linear relationship depends ONLY on the fact that Rao is computed directly from a Gower trait distance, while in FDis the Gower trait distance is first transformed into a PCoA and then the PCoA axes are used to compute the dissimilarities in the multivariate space. In some cases FEve is correlated negatively to FRic, as we discussed in Chapter 5 of the reference text book (see below for more details). We also see that Rao and FDis are increasing when the 'range' of traits, i.e. FRic (which is, as we discuss in Chapter 5 of the reference text book, is the size of the Convex Hull when multiple traits are considered, and the range, when only one trait is considered). We also see that both FDis and Rao are not correlated to the number of species. Finally, we also see that FDiv is quite correlated with FDis and Rao.

5.2.4.1 A slight detour

Before looking at the results obtained with `dbFD` we want to a make a small detour and show a couple of important things. First we would like to show how, for indices such as FDis, the Gower trait distance of the data is first transformed into a PCoA and then the PCoA axes are used to compute a new dissimilarity matrix. This is done automatically in PCoA but we want to show the overall process:

```
pcoNEspain <- dudi.pco(sqrt(gowdis(spxt[, 1:3])), scannf = FALSE, nf = 2)
```

In this case, for simplicity we retain only two axes of the PCoA space, using `scannf = FALSE` and nf = 2. If you want to see a similar example, see R material 2, on trait dissimilarity.

We can then look at the species scores on the PCoA, using the object `$li`:

```
head(pcoNEspain$li)
```

```
##                    A1           A2
## Acercamp  0.16765715  0.253227404
## Achimill -0.28923860  0.009246622
## Aegigeni  0.07643396  0.024944959
## Alchhybr -0.31491611  0.004550020
## Anemhepa -0.27591719  0.015519730
## Anthmont -0.31888517 -0.080193487
```

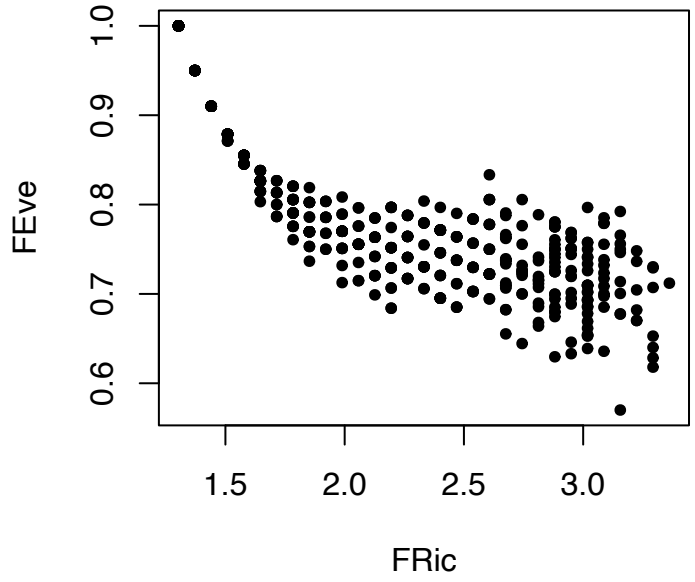
```
#these are the positions of the species in the first two axes of the PCoA analysis
```

Then we can compute an Euclidean distance with such an object and this distance will be used by indices such as FEve, FDis etc..

```
dist4FDis <- dist(pcoNEspain$li)
```

Second, we mentioned above (and in Chapter 5 of the reference text book) that FRic and FEve could be correlated between them, negatively, which is a undesired property (indices should be independent!). We show this using some relatively simple simulations. Of course if you are not very familiar with R you can simply skip this part and just see the resulting plot. If you are interested we simulated 500 plots, from a total of up to 50 species and a fixed species richness of 20 species per plot. A trait value between 1 and 50 is given to all species and a maximum trait range for the species in a plot is provided (notice that the range is actually FRic for one trait) with a random selection (minium range=20, max=50).

```
spxcomsim <- matrix(0, 500, 50) #create an empty matrix of 50 species, columns, and 500 plot
colnames(spxcomsim) <- 1:50 #give species names to the columns, 1 to 50
#simulate communities with variable trait range and fixed number of species:
for(i in 1:500){
  rangesim <- sample(20:50, 1) #creating a range of trait values between 20 and 50 for a plo
  nspsim <- 20 #create a fixed number of species for this plot, i.e. 20
  traitsim <- sample(1:rangesim, nspsim) #create species trait values within the trait range
  spxcomsim[i, traitsim] <- 1 #fill the spxcomsim for the species selected
}
#compute dbFD:
spxtsim <- as.matrix(1:50) #this creates a trait data for the 100 species, with a value from
rownames(spxtsim) <- 1:50 #we need the species names for dbFD to work
testsim <- dbFD(spxtsim, spxcomsim, message = F)
plot(testsim$FRic, testsim$FEve, pch = 20, xlab = "FRic", ylab = "FEve")
```

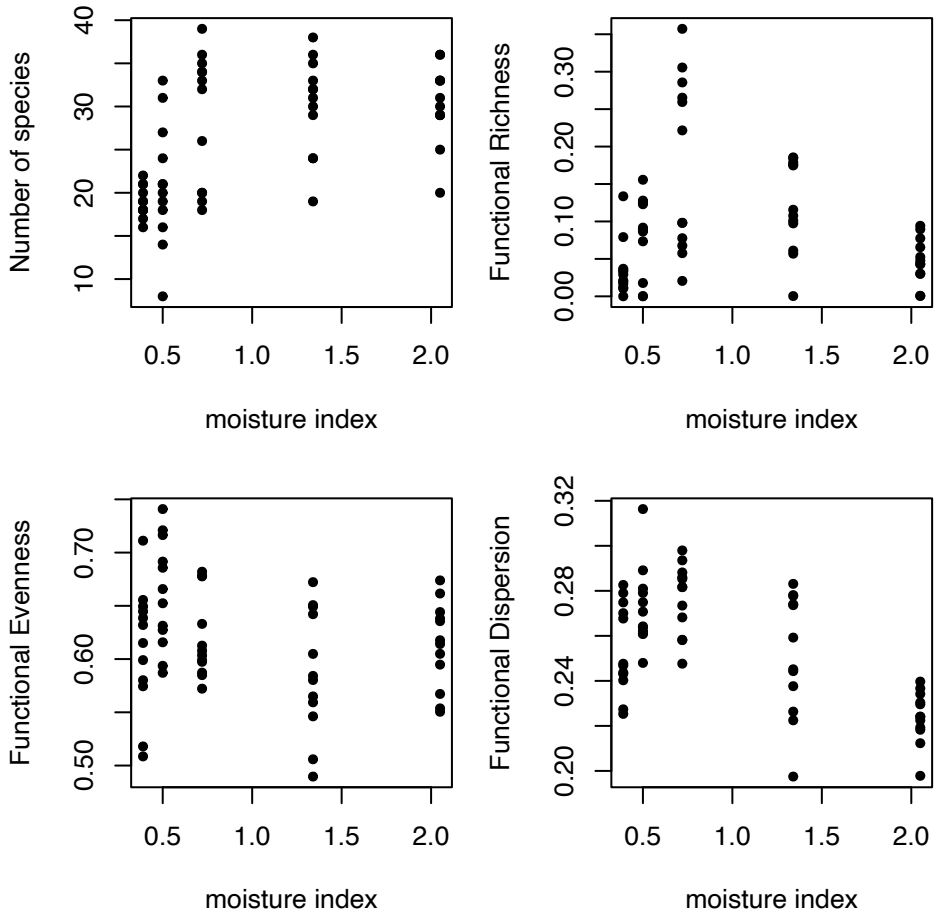


As we can see, for a given species richness value, the greater the trait range (i.e. FRic), the lower is FEve, i.e. species are more 'packed'. Of course this effect will be stronger when there are many species and the trait range will be relatively small. We show this effect to make the readers aware about the some limitations of this extisting FEve index. So we should suggest being careful when interpreting these FEve. An alternative has been proposed by Fontana et al. (2016).

5.2.4.2 Results with NE Spain data with dbFD

Having looked which indices are correlated between them allows us to select fewer indices for assessing the relationship between FD and, for example, the moisture index in the different vegetation belts in the NE Spain data

```
par(mfrow = c(2, 2)) #this is just to prepare the space for a figure composed by 4 plots, in a 2 b
par(mar = c(4, 4, 2, 1)) # margin of the plots, just to have things looking more or less good
plot(envxp$moisture.index, resFD$nbsp, xlab = "moisture index",
     ylab = "Number of species", pch = 20)
plot(envxp$moisture.index, resFD$FRic, xlab = "moisture index",
     ylab = "Functional Richness", pch = 20)
plot(envxp$moisture.index, resFD$FEve, xlab = "moisture index",
     ylab = "Functional Evenness", pch = 20)
plot(envxp$moisture.index, resFD$FDis, xlab = "moisture index",
     ylab = "Functional Dispersion", pch = 20)
```



What we see is that while the number of species is increasing with the moisture index, Functional Richness and Functional Dispersion tend to be maximized at lower or intermediate moisture sites. Hence *the mechanisms that maintain species diversity seems not to be the same supporting the functional differentiation between species.*

We can eventually apply different statistical tools to test the relationship with both the moisture index and grazing, as we did in the CWM exercise:

```
summary(lm(resFD$FDis ~ moisture.index * grazing, data = envxp))
```

```
##
## Call:
## lm(formula = resFD$FDis ~ moisture.index * grazing, data = envxp)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.043657 -0.013883  0.003567  0.014291  0.038247
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.268921   0.008024  33.514  < 2e-16 ***
## moisture.index          -0.020885   0.006821  -3.062  0.00338 **
## grazing                  0.011844   0.006216   1.905  0.06186 .
## moisture.index:grazing -0.004082   0.005284  -0.772  0.44307
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0207 on 56 degrees of freedom
```

```
## Multiple R-squared:  0.4149, Adjusted R-squared:  0.3836
## F-statistic: 13.24 on 3 and 56 DF,  p-value: 1.212e-06
```

The results show a general weak positive effect of grazing on some of the FD components.

We remind users (see Chapter 3 of the reference textbook, and R material Ch 3) that for computing FD with all traits considered in the 'spxt' matrix, which include *fuzzy coding ('LF_xx' labeled column)*, we need to complicate a bit more the script. Otherwise the Gower distance used in the dbFD function will 'understand' that each of the LF_xx column in the matrix is a different trait. We certainly do not want that, even if it would provide nicer results, as we now want the 4 traits to have the same weight and scale! We can use the approach already introduced for the R material Ch 3, i.e. compute the dissimilarity for each trait separately and then average the dissimilarity across traits:

```
head(spxt)
```

```
##          GrowhtForm LEG      SLA LF_Th LF_G LF_H LF_hCh LF_wCh LF_NP LF_P
## Acercamp      shrub   0 2.753661     0    0  0.0    0.0      0     0    1
## Achimill       forb   0 2.681022     0    0  0.5    0.5      0     0    0
## Aegigeni      grass   0 2.721295     1    0  0.0    0.0      0     0    0
## Alchhybr       forb   0 2.944439     0    0  1.0    0.0      0     0    0
## Anemhepa       forb   0 2.557227     0    0  1.0    0.0      0     0    0
## Anthmont       forb   1 2.602690     0    0  1.0    0.0      0     0    0
```

```
all.dist <- (gowdis(spxt["GrowhtForm"]) + gowdis(spxt["SLA"]) + gowdis(spxt["LEG"]) +
     gowdis(spxt[, 4:10]) / max(gowdis(spxt[, 4:10]))) / 4
resFD.alltraits <- dbFD(all.dist, log(t(spxp) + 1), message = F,
                        calc.CWM = FALSE, stand.FRic = TRUE)
summary(lm(resFD.alltraits$FDis ~ moisture.index * grazing, data = envxp))
```

```
##
## Call:
## lm(formula = resFD.alltraits$FDis ~ moisture.index * grazing,
##     data = envxp)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.040506 -0.009844  0.003573  0.013028  0.034711
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.315676   0.007360  42.891  < 2e-16 ***
## moisture.index           -0.024385   0.006257  -3.897 0.000262 ***
## grazing                   0.015454   0.005701   2.711 0.008895 **
## moisture.index:grazing -0.014362   0.004846  -2.963 0.004460 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01899 on 56 degrees of freedom
## Multiple R-squared:  0.6517, Adjusted R-squared:  0.6331
## F-statistic: 34.93 on 3 and 56 DF,  p-value: 7.358e-13
```

The new results are quite different, and including Life Form improves the predictions of the model.

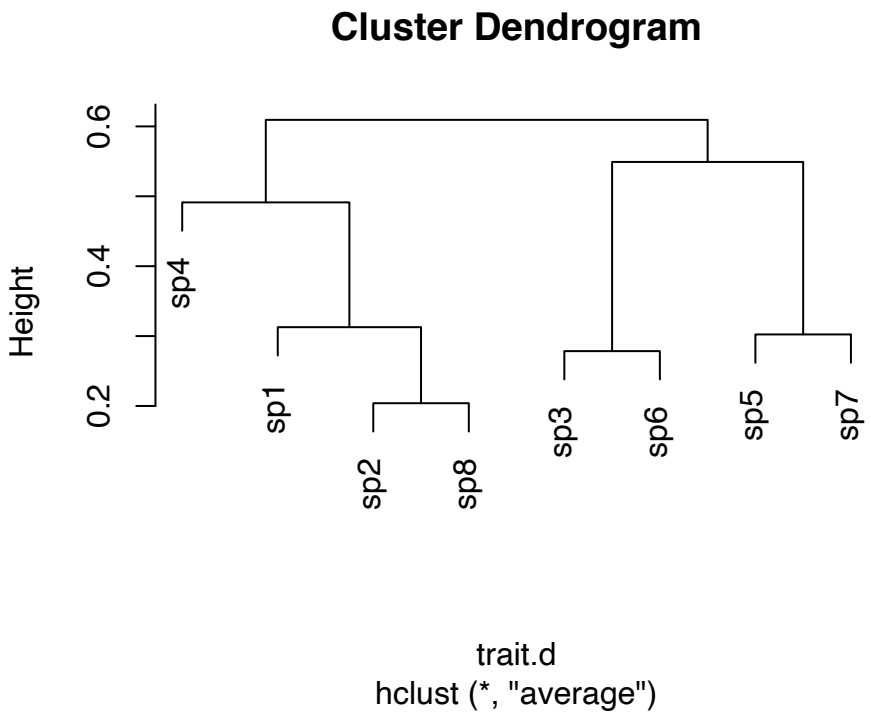
5.2.5 Package `picante`

We will now focus on other FD indices that can be computed with the package *picante*. First we will be using the function `pd` which computes Faith's Diversity, which is identical to the index proposed by Petchey and Gaston (2002). Then we will focus on the functions `mpd` and `mntd`, while introducing another function 'melodic', developed to overcome unexpected results with the function `mpd` (see de Bello et al., 2016).

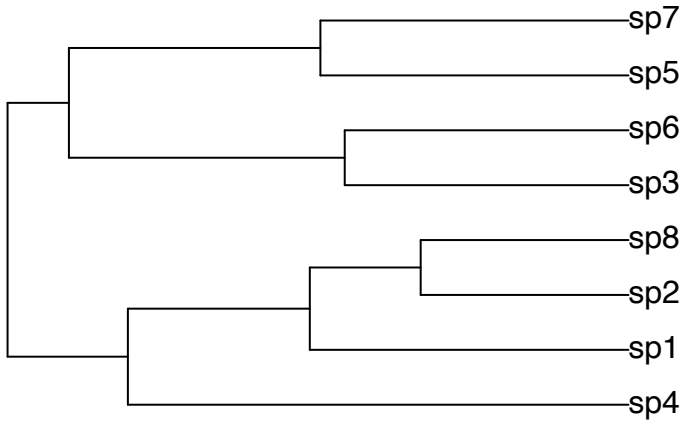
5.2.5.1 The function `pd`

Let's try first with the simple data 'dummy' already introduced above. We can calculate once again the Gower distance with the `dummy$trait` data as example, and then we compute a *dendrogram*. Such dendrogram can be made in many possible ways, and discussions have been published on this topic (see for example Mouchet et al., 2008). Here, for simplicity, we just use a classic hierarchical clustering with the function `hclust`, similarly to what we used in the R material Ch 3. Here are the steps:

```
library(picante)
trait.d <- gowdis(dummy$trait)
tree.traits <- hclust(trait.d, "average")
plot(tree.traits)
```



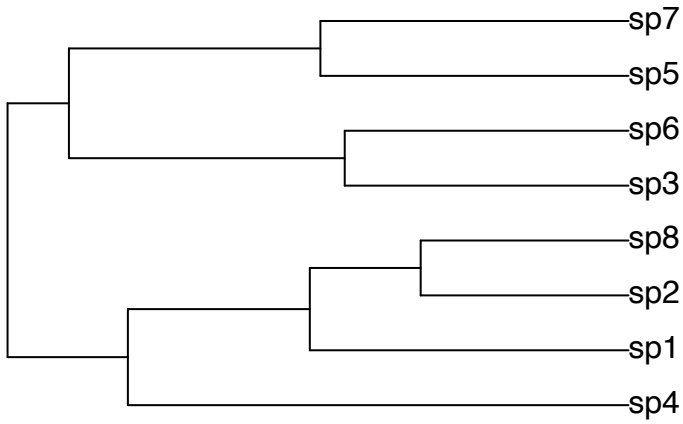

```
plot(as.phylo(tree.traits))
```



It looks as the species 5 and 7 are very similar between them and very different from species 4 (i.e. the *lenght of the branches, which express dissimilarity*, to connect species 7 and 4 is long).

We also show that we can transform the tree so that all branches end at the same level, using `as.phylo()` This basically means to transform the tree into an *ultrametric tree* (a tree where all the path-lengths from the root to the tips are equal). Just to make this clear, let's create a new object, with such an ultrametric tree:

```
ultra.tree.traits <- as.phylo(tree.traits)
plot(ultra.tree.traits)
```

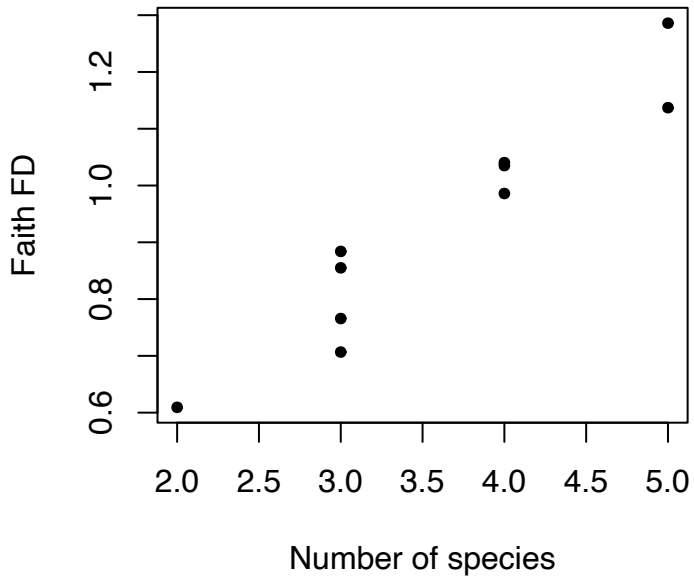


We are ready to compute functional diversity. When using the `pd` function we need to introduce the Species x communities matrix first (in this case `dummy$abun`, i.e. with species as columns) and the tree built on the traits:

```
faithFD <- pd(dummy$abun, ultra.tree.traits)
faithFD #notice that the functional diversity values are contained in the column PD, because the j
```

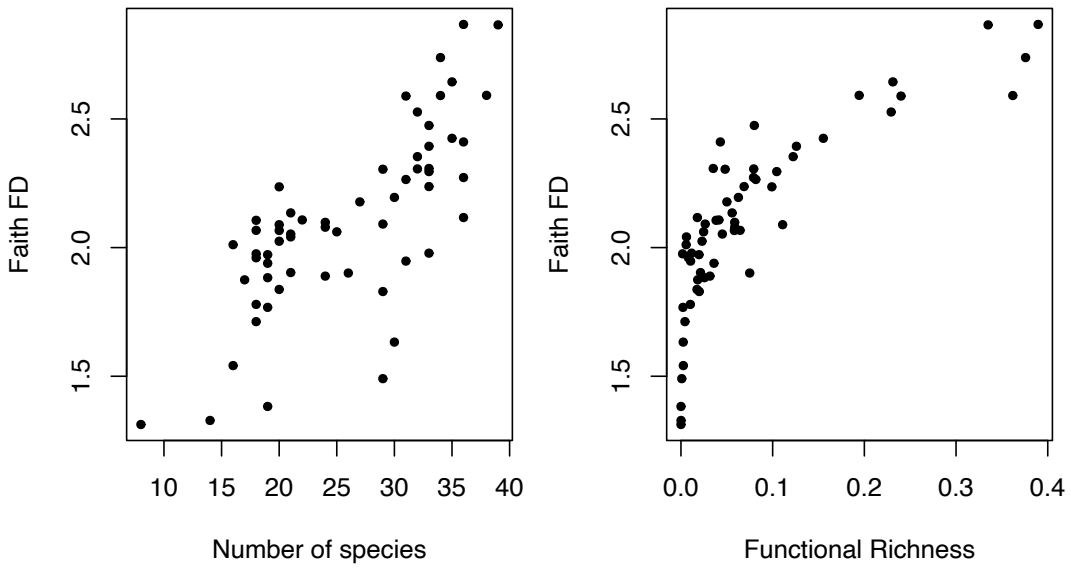
```
##               PD SR
## com1  1.0403074  4
## com2  0.8550039  3
## com3  0.7657313  3
## com4  0.6093634  2
## com5  0.8839394  3
## com6  1.1370824  5
## com7  0.7066902  3
## com8  1.0351102  4
## com9  1.2859479  5
## com10 0.9859116  4
```

```
plot(faithFD$SR, faithFD$PD, pch = 20, ylab = "Faith FD",
     xlab = "Number of species")
```



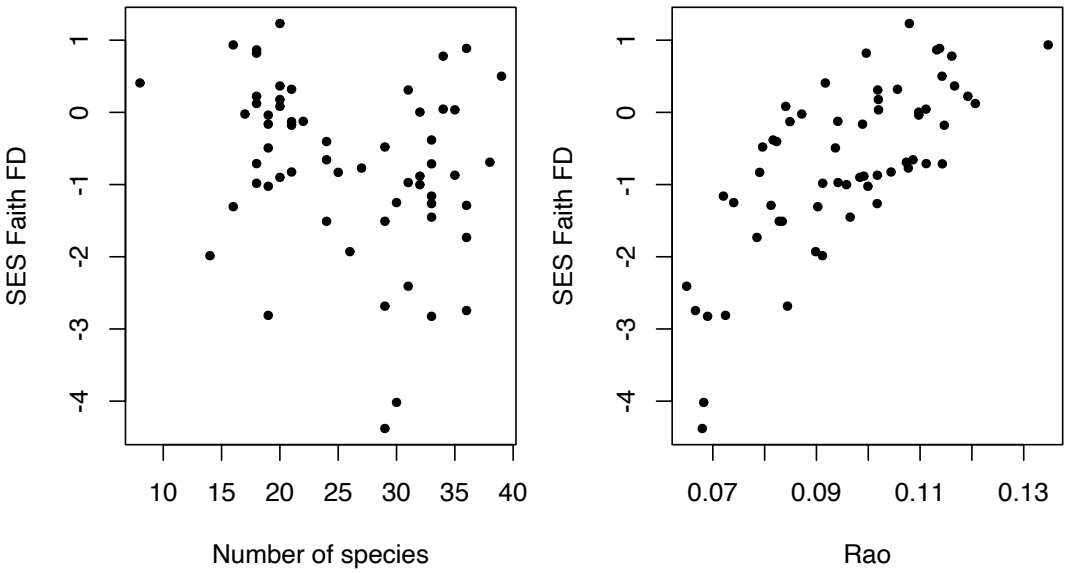
As you can see the index is very much correlated with the number of species. This is even more clear when using a bigger dataset, such as the NE Spain data already used above. We also now relate this index to the Functional Richness computed above. Let's start by creating the classic hierarchical clustering (hclust) based on `all.dist` which is the dissimilarity matrix between all traits in NE Spain dataset:

```
tree.traits.NEspain <- hclust(all.dist, "average")
ultra.tree.traitsNEspain <- as.phylo(tree.traits.NEspain)
faithFD.NEspain <- pd(t(spxp), ultra.tree.traitsNEspain)
resFD.alltraits<-dbFD(all.dist, log(t(spxp) + 1), message = F, calc.CWM = FALSE,
                      stand.FRic = TRUE) # let's run this command again, as we used above, t
par(mfrow=c(1,2))
par(mar = c(4, 4, 2, 1))
plot(faithFD.NEspain$SR, faithFD.NEspain$PD, pch=20, ylab="Faith FD", xlab="Number of specie
plot(resFD.alltraits$FRic, faithFD.NEspain$PD, pch=20, ylab="Faith FD", xlab="Functional Ric
```



Can we thus use these indices to measure Functional Diversity? They provide an information which is very much redundant with the number of species. If we want to 'remove' the inherent effect of species richness on this index, and assess their information beside the effect of species richness also, we can use randomizations. As we will see in Chapter 7 of the reference textbook, and R material Ch 7, we can use *randomizations* to 'remove' the inherent effect of species richness on these FD indices. This is very simple with the function `ses.pd` already existing in 'picante'. If used by default, the function will randomize species names (as we did in the exercise with CWM) and use 999 randomizations. We will do a quicker test here with only 199 randomizations (with the argument `runs`), just as an exercise. The function is called `ses.pd`, referring to the *standardized effect size* (see Chapter 7, R material 7) of pd. These values however are stored with the name `pd.obs.z`. The number of species is stored in a column name called `ntaxa`.

```
ses.faithFD.NEspain <- ses.pd(t(spxp), ultra.tree.traitsNEspain, runs = 199)
par(mfrow = c(1, 2))
par(mar = c(4, 4, 2, 1))
plot(ses.faithFD.NEspain$ntaxa, ses.faithFD.NEspain$pd.obs.z, pch = 20,
     ylab = "SES Faith FD", xlab = "Number of species")
plot(resFD.alltraits$RaoQ, ses.faithFD.NEspain$pd.obs.z, pch = 20,
     ylab = "SES Faith FD", xlab = "Rao")
```



We now see that the Faith index is not correlated any more to the number of species, but it generally reflects quite well the Rao index computed above.

5.2.5.2 The functions 'mpd', 'mntd' and 'melodic' for MPD and Rao

Let's now see two very popular indices of FD (and also of phylogenetic diversity), *MPD* and *MNTD*. The two respective functions in the package 'picante' (i.e. `mpd` and `mntd`) works in the same way: you need to introduce a species x communities matrix (such as dummy$abun, where species are columns) and a dissimilarity object. This latter one, differently from other functions used so far, is not a 'triangle' anymore, but it is a *full matrix, with two symmetric triangles*, for example:

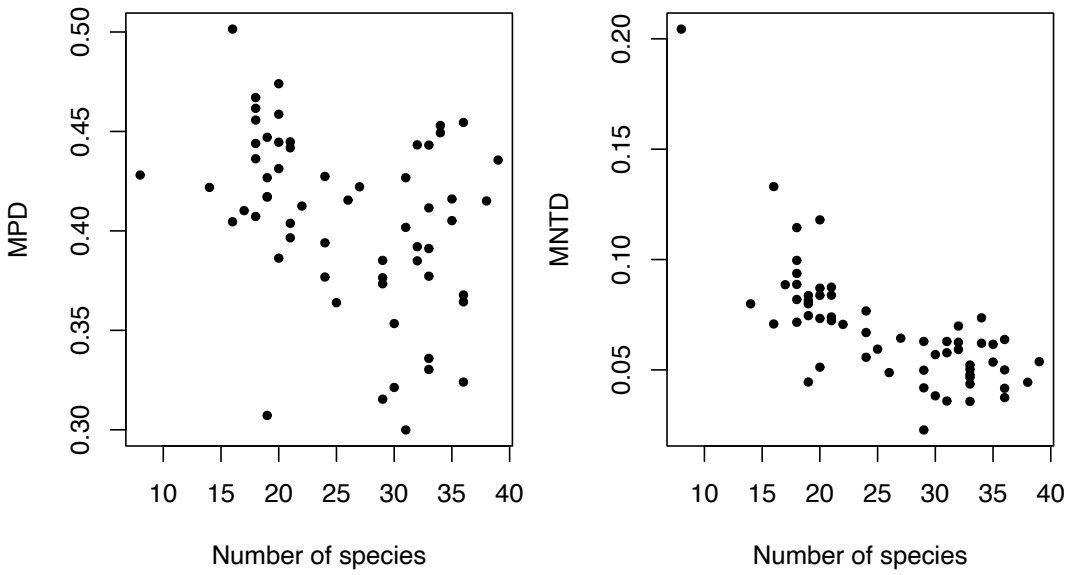
```
round(as.matrix(trait.d), 3)
```

```
##       sp1   sp2   sp3   sp4   sp5   sp6   sp7   sp8
```

```
## sp1 0.000 0.218 0.524 0.674 0.529 0.610 0.448 0.407
## sp2 0.218 0.000 0.668 0.561 0.815 0.593 0.686 0.204
## sp3 0.524 0.668 0.000 0.823 0.486 0.278 0.485 0.596
## sp4 0.674 0.561 0.823 0.000 0.484 0.707 0.558 0.239
## sp5 0.529 0.815 0.486 0.484 0.000 0.607 0.302 0.559
## sp6 0.610 0.593 0.278 0.707 0.607 0.000 0.619 0.447
## sp7 0.448 0.686 0.485 0.558 0.302 0.619 0.000 0.703
## sp8 0.407 0.204 0.596 0.239 0.559 0.447 0.703 0.000
```

Let's now try to use it directly with the NE Spain data:

```
mpd.NEspain <- mpd(t(spxp), as.matrix(all.dist))
mntd.NEspain <- mntd(t(spxp), as.matrix(all.dist))
number.species <- faithFD.NEspain$SR # we take the number of species from one of the most re
par(mfrow = c(1, 2))
par(mar = c(4, 4, 2, 1))
plot(number.species, mpd.NEspain, pch = 20, ylab = "MPD",
     xlab = "Number of species")
plot(number.species, mntd.NEspain, pch = 20, ylab = "MNTD",
     xlab = "Number of species")
```



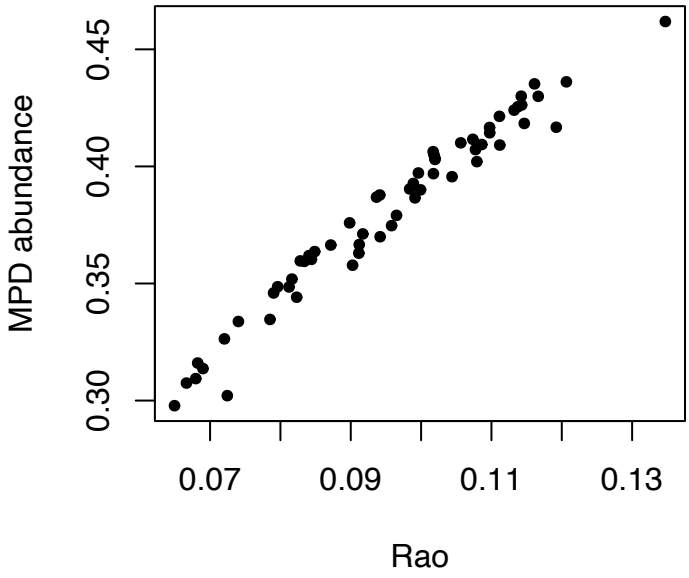
We observe a general lack of correlation between MPD and the number of species (although actually in this case Pearson R was -0.36). As the same time, as expected, we observe a negative correlation between MNTD and the number of species (see Chapter 5 in the reference book).

Up to now we have computed MPD and MNTD without considering species abundances. *Species abundances* can be considered with the argument `abundance.weighted=TRUE`. This is apparently, and only apparently, simple. Let's try. Remember that above, for the NE Spain data, we log transformed the abundance data, to damp a bit the effect of overabundant species, so we do it here as well.

```
mpd.NEspain.ab <-mpd(log(t(spxp) + 1), as.matrix(all.dist),
                     abundance.weighted = TRUE)
mntd.NEspain.ab <- mntd(log(t(spxp) + 1), as.matrix(all.dist),
                        abundance.weighted = TRUE)
```

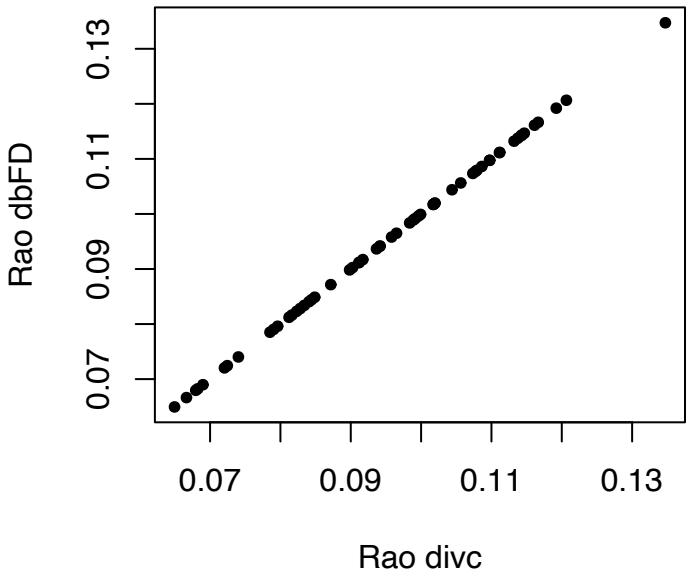
Simple, right? Yes! For MNTD everything seems ok, but actually *not for MPD*!! This is the relationship between MPD and Rao computed so far.

```
plot(resFD.alltraits$RaoQ, mpd.NEspain.ab, pch = 20, ylab = "MPD abundance",
     xlab = "Rao")
```



First let's consider Rao. We assume that the dbFD function computes Rao with the formula given by Botta-Dukát (2005) and many others. However, actually, the dbFD function uses the function `divc`, to compute RaoQ, from the package `ade4`, which works like this:

```
Rao.divc <- divc(log(spxp + 1), all.dist)
plot(Rao.divc$diversity, resFD.alltraits$RaoQ, ylab = "Rao dbFD",
     xlab = "Rao divc", pch = 20)
```

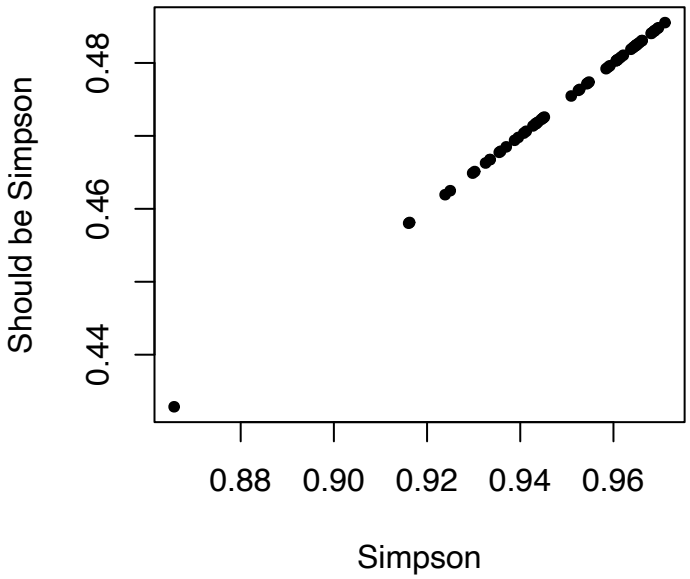


Yes, we just confirmed that the dbFD function uses `divc`. The 'problem' here is that in both functions, when computing Rao, the dissimilarity matrix is squared and divided by 2. This is NOT the formula given by Botta-Dukát (2005). This approach is expressing Rao as a measure *variance* (see de Bello et al., 2011). In principle this is not a problem and

results produced with the 'real' Rao and with Rao expressed as variance will be very tightly correlated. But it needs some clarification.

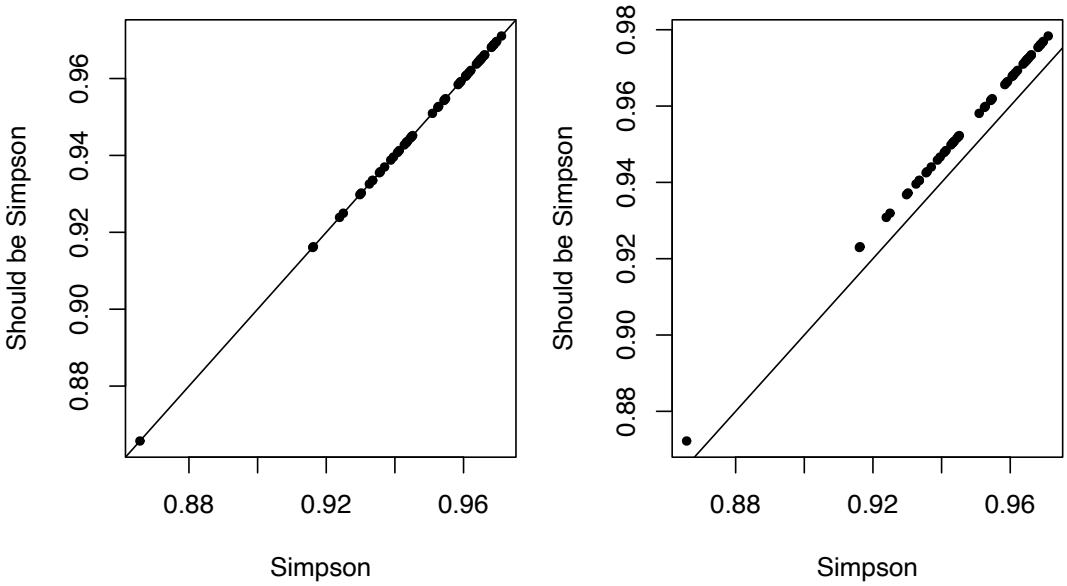
Let's first show that the function `divc` does not work exactly as one could expect. An important expectation, if we consider Botta-Dukát (2005), is that *when the dissimilarity between each pair or species is equal to 1, then the Rao index should be the same as Simpson diversity.* Let's try this:

```
Simpson <- diversity(log(t(spxp) + 1), index = "simpson") # using the diversity function in
dist.all1 <- as.dist(matrix(1, dim(spxp)[1], dim(spxp)[1]))# creating a matrix of dissimilar
shouldbeSimpson <- divc(log(spxp + 1), dist.all1)
plot(Simpson, shouldbeSimpson$diversity, xlab = "Simpson",
     ylab = "Should be Simpson", pch = 20)
```



Yes, they are *perfectly correlated but the values in the Y axis are definitely NOT the same as in the X axis.* Therefore the function `divc` does not produce the Rao index, at least not in the form of Botta-Dukát (2005) referred in the manual of dbFD. We can have a 'home made' solutions or use one with the argument 'scale' in the `divc` function (and 'scale.RaoQ' in dbFD). Only 'home made' solutions really returns the expected values (i.e. results are the same Simpson when the dissimilarity is equal to 1 between all species pairs)

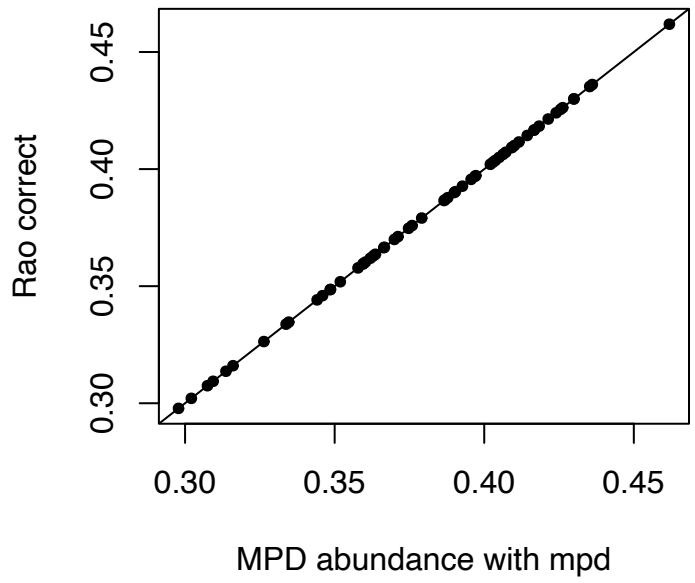
```
shouldbeSimpson.correction1 <- divc(log(spxp + 1), sqrt(dist.all1) * 2) / 2
shouldbeSimpson.correction2 <- divc(log(spxp + 1), dist.all1, scale = T)
par(mfrow = c(1, 2))
par(mar = c(4, 4, 2, 1))
plot(Simpson, shouldbeSimpson.correction1$diversity, xlab = "Simpson",
     ylab = "Should be Simpson", pch = 20)
abline(0, 1) #this is the 1:1 line
plot(Simpson, shouldbeSimpson.correction2$diversity, xlab = "Simpson",
     ylab = "Should be Simpson", pch = 20)
abline(0, 1)
```



As you can see only the first approach, `shouldbeSimpson.correction1`, really returns values equal to Simpson. You might ask: why all this is important? The take home messages are (1) that `divc` and dbFD computes a form of Rao which is variance and (2) this does not scale well with Simpson (in the case that the dissimilarity is equal to 1 for all species pair).

Then, assuming that the approach used to compute `shouldbeSimpson.correction1` is what we need, we can compare the results with the one using MPD with species abundance. Here comes the *surprise*:

```
Rao.NEspain.correct <- divc(log(spxp + 1), sqrt(all.dist) * 2) / 2 #this is Rao that scales correc
mpd.NEspain.ab <- mpd(log(t(spxp) + 1), as.matrix(all.dist), abundance.weighted = TRUE)
plot(mpd.NEspain.ab, Rao.NEspain.correct$diversity,
     xlab = "MPD abundance with mpd", ylab = "Rao correct", pch = 20)
abline(0, 1)
```



Nice. What does it mean? Does it mean that MPD is the same as Rao? Some have argued that the weighted version of MPD is Rao (for example Swenson, 2014). We are fine with this view. The problem is that we think this view can be, possibly, improved. There is actually another abundance version MPD, as we described in de Bello et al. (2016), also following

earlier works. In summary, as discussed in the reference textbook, Chapter 5, MPD and Rao are similar indices, but are not the same.

As a matter of fact we do still not understand why the 'mpd' function computes the things in different ways when using abundance or not. When `abundance.weighted=TRUE` the `mpd` function considers only the dissimilarity between different species, i.e. not the species with it self, which is the diagonal. So in this case the diagonal of the dissimilarity matrix is discarded (whose values are equal to 0). On the contrary when `abundance.weighted=FALSE` the `mpd` function considers the diagonal. We expect the authors believe the view that the weighted version of MPD is Rao. If so, by computing a weighted version of MPD, which does not consider the diagonal, then we obtain a nice relationship which can be summarized as *MPD=Rao/Simpson*.

To demonstrate these issues and solve problems, we made a function `melodic` (for an harmonious use of these indices) which does this. It works as `mpd` and computes both weighted and unweighted forms of MPD and Rao.

```
source(here::here("data", "chapter5", "melodic.r"))
res.melodic.NESpain <- melodic(log(t(spxp) + 1), as.matrix(all.dist))
str(res.melodic.NESpain)
```

```
## List of 3
##  $ abundance:List of 3
##   ..$ simpson: num [1:60] 0.93 0.945 0.943 0.943 0.945 ...
##   ..$ rao    : num [1:60] 0.366 0.388 0.414 0.39 0.41 ...
##   ..$ mpd    : num [1:60] 0.394 0.41 0.439 0.414 0.434 ...
##  $ presence :List of 3
##   ..$ simpson: num [1:60] 0.941 0.955 0.947 0.95 0.952 ...
##   ..$ rao    : num [1:60] 0.386 0.394 0.424 0.41 0.424 ...
##   ..$ mpd    : num [1:60] 0.41 0.412 0.447 0.431 0.445 ...
##  $ richness : int [1:60] 17 22 19 20 21 19 18 18 21 18 ...
```

```
## is MPD=Rao/Simpson, for both the presence/absence and abundance forms?
res.melodic.NESpain$abundance$mpd == res.melodic.NESpain$abundance$rao /
  res.melodic.NESpain$abundance$simpson #yes, with the abundance form!
```

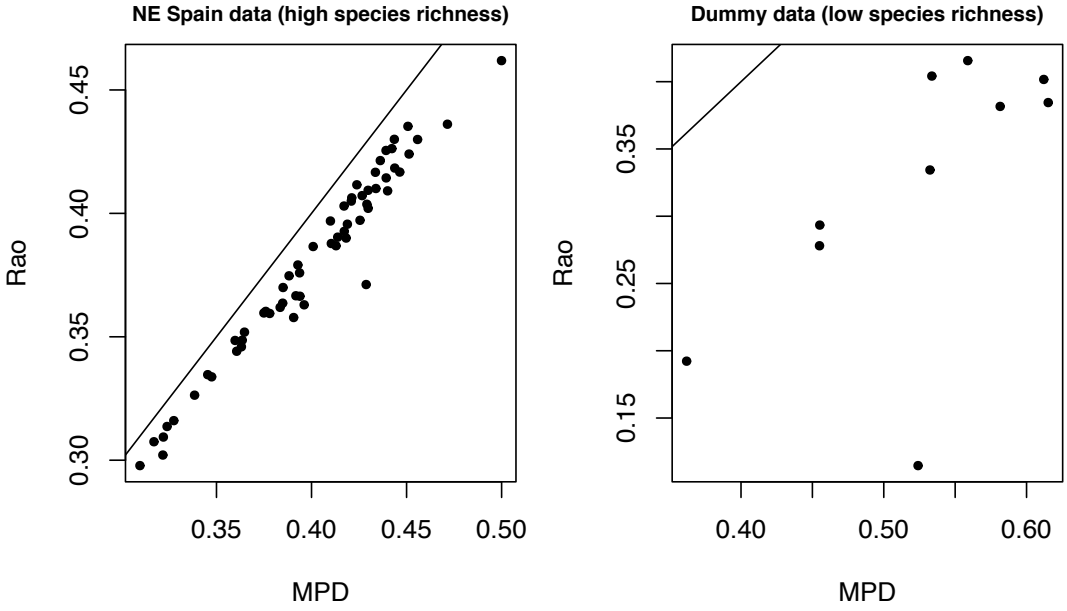
```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
res.melodic.NESpain$presence$mpd == res.melodic.NESpain$presence$rao /
  res.melodic.NESpain$presence$simpson #yes, with the presence/absence form!
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
res.melodic.dummy <- melodic(dummy$abun, as.matrix(trait.d))
par(mfrow = c(1, 2))
par(mar = c(4, 4, 2, 1))
plot(res.melodic.NESpain$abundance$mpd, res.melodic.NESpain$abundance$rao,
     xlab = "MPD", ylab = "Rao", pch = 20, main = "NE Spain data (high species richness)",
     cex.main = 0.8)
abline(0, 1)
plot(res.melodic.dummy$abundance$mpd, res.melodic.dummy$abundance$rao ,
```

```
     xlab = "MPD", ylab = "Rao", pch = 20, main = "Dummy data (low species richness)",
     cex.main = 0.8)
abline(0, 1)
```



In practice, especially when *communities are species rich* (say, e.g., more that 15-20 species) there will be basically *no difference between MPD and Rao* (figure above, NE Spain data). But when the *communities are relatively species poor* there will be *difference between the weighted form of Rao and the ('real') weighted form of MPD* (figure above, dummy data). Moreover, as demonstrated by Ricotta et al. (2016), the form *1-Rao/Simpson* (which is the *1-MPD*, with the MPD form computed in 'melodic') represents a version of *functional redundancy*. In fact it express how much of the maximum dissimilarity possible in a community (Simpson, i.e. the dissimilarity between all species pairs is equal to 1) is represented by the observed functional diversity (Rao). If Rao==Simpson -> MPD==1, then there is no redundancy. The lower Rao with respect to Simpson, the higher is redundancy.

5.2.6 Alpha, Beta and Gamma FD

In Chapter 5 we also discuss options to *partition functional diversity*, in a way similar to species diversity, i.e. in alpha and beta diversity. The literature about biodiversity partitioning is generally very complex, often with polarized views (e.g., Jost, 2007). The partitioning of functional diversity has not been solved for many indices. *The most reliable one seems Rao*, following the work by de Bello et al. (2010a) and Botta-Dukát (2018). Other options involve computing convex hull, similar to the FRic index, and look at intersections of the convex hull between communities. This is done in the package betapart (https://cran.r-project.org/web/packages/betapart/betapart.pdf) and using the functions `functional.beta.multi` and `functional.beta.pair`. We will introduce similar indice in the R material Ch 6. Here we focus instead mainly on Rao, following the reference textbook Chapter 5 and based on the fact that this approach can also consider species abundances.

There has been a debate, in Journal of Ecology, between Villeger & Mouillot (Villéger and Mouillot, 2008) and Hardy and Jost (2008), on how Rao should be expressed, which is summarized for non mathematicians in the work by de Bello et al. (2010a) . This work proposes solutions that solves the debate mentioned above and applies the corrections proposed by Jost for species diversity indices such as Simpson. All these concepts are resolved in the R function `Rao` (de Bello et al., 2010a), which is at the moment the only available function

solving these issues. We can open the function (which actually can also account for phylogentic distance between species) and run it with few examples. Notice that in this case the 'species x communities' matrix has species as rows. We can first play with the simple example provided in Chapter 5 of the reference textbook, which we recreate below:

```
source(here::here("data", "chapter5", "Rao.r"))
comm.1 <- c(1, 1, 0, 0, 0)
comm.2 <- c(0, 0, 1, 1, 0)
comm.3 <- c(0, 0, 1, 1, 1)
coms <- cbind(comm.1, comm.2, comm.3)
rownames(coms) <- paste("sp", 1:5, sep = ".")
coms
```

```
##      comm.1 comm.2 comm.3
## sp.1      1      0      0
## sp.2      1      0      0
## sp.3      0      1      1
## sp.4      0      1      1
## sp.5      0      0      1
```

```
trait.reg1 <- c("grass", "grass", "forb", "forb", "grass")
trait.reg2 <- c("grass", "grass", "legume", "forb", "grass")
partRao.Reg1 <- Rao(coms, dfunc = gowdis(as.data.frame(trait.reg1)), dphyl = NULL,
                    weight = F, Jost = T, structure = NULL)
partRao.Reg2 <- Rao(coms, dfunc = gowdis(as.data.frame(trait.reg2)), dphyl = NULL,
                    weight = F, Jost = T, structure = NULL)
```

The Rao function needs a species x communities matrix with species as rows and does not need that the dissimilarity matrix will include species names (it assumes, maybe wrongly, users are careful enough to have species in the same order in the trait matrix and in the species composition matrix). We just run the calculations, using the Jost correction (Jost=T) to avoid having an underestimation of beta diversity (see Chapter 5 in the reference textbook for more details). We can now find the results used which we need to build a figure similar to Fig.5.6 in the reference textbook. The alpha taxonomic diversity (which is the mean Simpson per plot), the beta and gamma, can be obtained as:

```
partRao.Reg1$TD$Mean_Alpha #alpha
```

```
## [1] 2.25
```

```
partRao.Reg1$TD$Beta_add #beta
```

```
## [1] 2.25
```

```
partRao.Reg1$TD$Mean_Alpha + partRao.Reg1$TD$Beta_add == partRao.Reg1$TD$Gamma #alpha+beta=g
```

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

Indeed alpha + beta = gamma. We are aware some authors do not like to express beta diversity in additivity terms, so it has to be expressed in proportional terms, for example.

```
partRao.Reg1$TD$Beta_add * 100 / partRao.Reg1$TD$Gamma #beta / gamma
```

```
## [1] 50
```

```
partRao.Reg1$TD$Beta_prop #which is the same as above
```

```
## [1] 50
```

You can apply the same procedure between each pair of plot. The taxonomical beta diversity between each pair of plots is shown in the following object. Notice that between a pair of plots the value 50% implies a complete functional replacement/turnover (see textbook for more details). In fact, the value obtained, i.e. 'Beta_prop', depend on the number of plots considered (the maximum turnover is 50% for two plots and increases with increased number of plots). As such 'Beta_prop' can be standardized, to be independent on the number of plots, with the formula 14 in de Bello et al. (2010a).

```
round(partRao.Reg1$TD$Pairwise_samples$Beta_prop) #value obtained without standardization
```

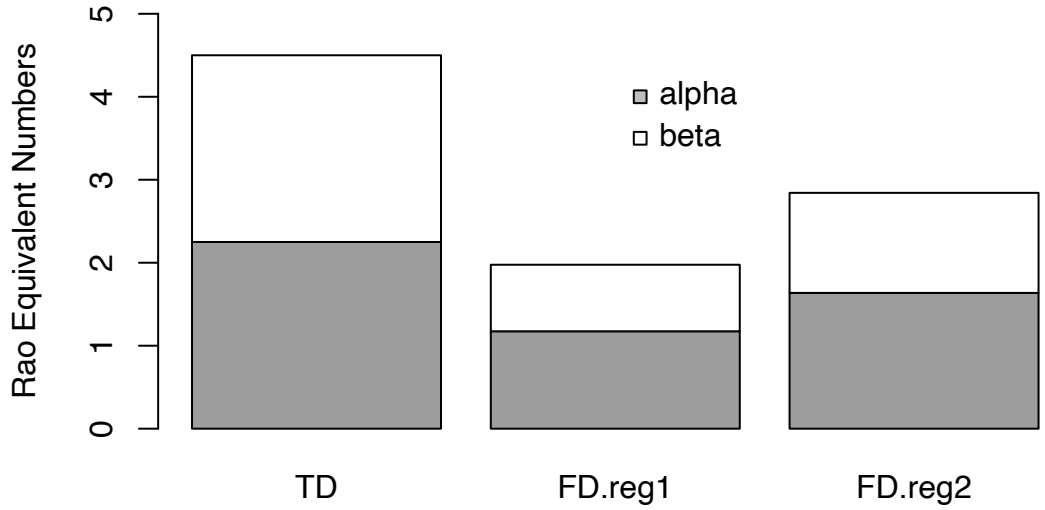
```
##        comm.1 comm.2
## comm.2     50
## comm.3     50     10
```

```
round(partRao.Reg1$TD$Pairwise_samples$Beta_prop / (1 - 1 / 2)) #standardized value, where 2 is th
```

```
##        comm.1 comm.2
## comm.2    100
## comm.3    100     20
```

We can now try to make a figure similar to Fig.5.6.

```
TD <- c(partRao.Reg1$TD$Mean_Alpha, partRao.Reg1$TD$Beta_add)
FD.reg1 <- c(partRao.Reg1$FD$Mean_Alpha, partRao.Reg1$FD$Beta_add)
FD.reg2 <- c(partRao.Reg2$FD$Mean_Alpha, partRao.Reg2$FD$Beta_add)
barplot(cbind(TD, FD.reg1, FD.reg2), beside = F, col = c(8, 0),
        ylim = c(0, 5), ylab = c("Rao Equivalent Numbers"))
points(2, 4, pch = 22, bg = "grey")
points(2, 3.5, pch = 22, bg = "white")
text(2, 4, "alpha", pos = 4)
text(2, 3.5, "beta", pos = 4)
```



In practice we can see that in the second region (or in general when considering the trait 'trait.reg2') we have higher alpha but also higher beta diversity (i.e. functional replacement). As a proportion, however, the functional replacement is the same in both regions (i.e., ~42% of the trait diversity is between plots, as compared to within plots):

```
partRao.Reg1$FD$Beta_prop
```

```
## [1] 40.57971
```

```
partRao.Reg2$FD$Beta_prop
```

```
## [1] 42.42424
```

Let's now work with the NE Spain data:

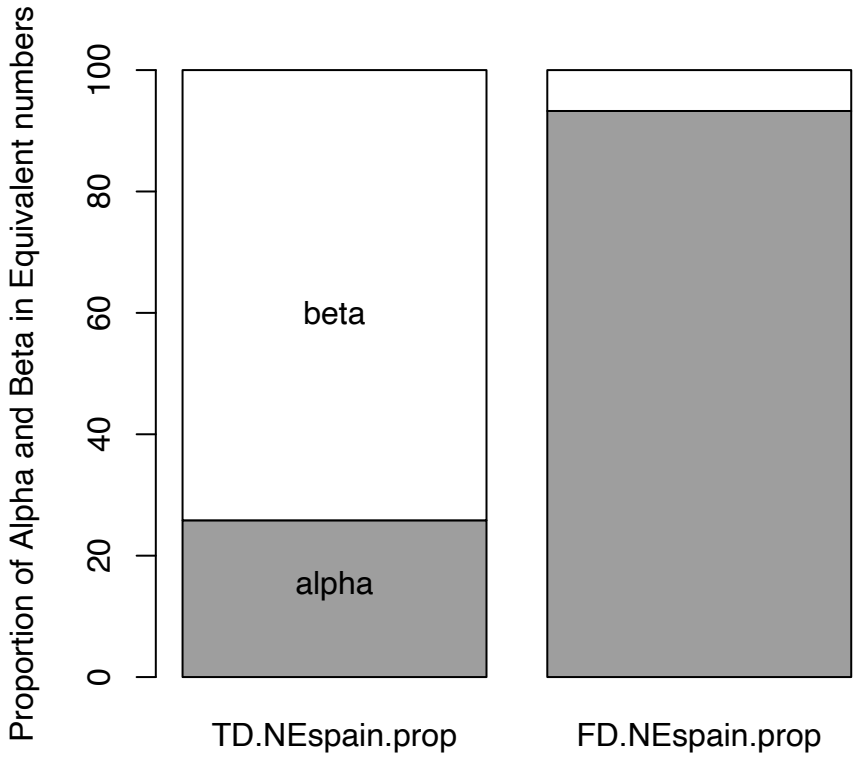
```
partRao.NESpain <- Rao(log(spxp + 1), all.dist, dphyl = NULL, weight = F, Jost = T,
                       structure = NULL)
partRao.NESpain$TD$Beta_prop
```

```
## [1] 74.17581
```

```
partRao.NESpain$FD$Beta_prop
```

```
## [1] 6.737896
```

```
TD.NEspain.prop <- c(100 * partRao.NESpain$TD$Mean_Alpha / partRao.NESpain$TD$Gamma,
                     partRao.NESpain$TD$Beta_prop)
FD.NEspain.prop <- c(100*partRao.NESpain$FD$Mean_Alpha/partRao.NESpain$FD$Gamma,
                     partRao.NESpain$FD$Beta_prop)
barplot(cbind(TD.NEspain.prop, FD.NEspain.prop), beside = F, col = c(8, 0),
        ylim = c(0, 100), ylab = c("Proportion of Alpha and Beta in Equivalent numbers"))
text(0.7, 60, "beta")
text(0.7, 15, "alpha")
```



We can see in this figure that, *despite the great turnover in species composition* (beta TD, grey colour, left stack) we have a real *low functional turnover*. In other words **most of the trait dissimilarity between species is found within a plot** and not across plots, even if there are such a marked environmental changes and even if species composition changes are very strong (high beta TD).

6 – Intraspecific trait variability

Here, we will explore how to estimate the contribution of *intraspecific trait variability (ITV)* to the total trait variability that we observe both *within communities* and *among communities*. How can we quantify the contribution of ITV to total trait variability in these two cases? ITV within communities can be important when the differences in trait values among the individuals of a species are relatively large compared to the differences between the average trait values of species. When studying ITV among communities, we are interested in finding out how much of the change in trait values along an environmental gradient is due to intraspecific adjustments of species and how much is due to species replacement. Here, we will introduce methods to estimate these contributions of ITV to total trait variability. Then, we will learn how to use methods that incorporate ITV into estimations of species dissimilarities and different indices of functional diversity. Most of the examples shown here correspond to Chapter 6 in the reference book, although the parts about estimating functional dissimilarity and different indices of functional diversity while considering ITV are also discussed in Chapter 3 and Chapter 5.

6.1 Data

In all this section we will use data from Carmona et al. (Carmona et al., 2015b). In their paper, the authors sampled Mediterranean annual grasslands ca. 20 km north of Madrid, in central Spain. They selected a topographical gradient encompassing substantial differences in vegetation productivity within a limited spatial scale. The gradient runs 80 m downhill from the upper part of a slope with a steep inclination. There are substantial differences in soil water content, clay percentage, total nitrogen and soil organic matter content associated with the position along the slope, which are finally reflected in a much higher productivity of down-slope areas. The authors placed 40 20cm x 20cm quadrats in a line, at distances of 2 m from each other. In each quadrat, they recorded plant species cover, and collected 10 individuals of the most abundant species (those that accounted for at least 90% of the cover of the quadrat). For these selected individuals they measured some functional traits, including plant height and specific leaf area (SLA), which are the traits we will consider here. This sampling strategy gives data at the "population" level (species within quadrats, for a total of 254 populations of 51 species). For this exercise we will consider a subset of these, including 10 plots sampled at equal intervals along the gradient.

Let's load the data and explore them a bit:

```
traits <- read.table(here::here("data","chapter6", "Traits.txt"), header=T, stringsAsFactors
commMatrix <- read.table(here::here("data","chapter6","CommMatrix.txt"), header=T, stringsAs
```

The object `traits` contains the Height and SLA values of each individual in each plot (10 plots, which are indicated in the `Plot` column), as well as the species to which individuals belong (10 individuals per species per plot)

```
summary(traits)
```

```
##       Plot          Species        Height           SLA
##  Min.   : 1.00   Pla_lag:100   Min.   : 1.00   Min.   : 6.886
##  1st Qu.:10.00   Tri_che: 77   1st Qu.: 5.30   1st Qu.:20.204
##  Median :23.00   Bis_pel: 60   Median : 9.90   Median :24.811
```

```
##  Mean   :21.52   Ech_pla: 49   Mean   :11.87   Mean   :25.858
##  3rd Qu.:31.00   Ant_arv: 40   3rd Qu.:15.78   3rd Qu.:30.282
##  Max.   :40.00   Tol_bar: 30   Max.   :49.00   Max.   :65.192
##                  (Other):260
```

The object `commMatrix` contains the relative abundance of the species. While raw measurements of cover can be more than 100% (when different plants overlap) or less than 100% (when parts of the plot are occupied by things other than vegetation, such as bare soil), the relative abundance of a species in a plot reflect what proportion of the total plant cover belongs to that species. This means that the abundances of all species (columns) in a plot (rows) should add up to 1. Let's check that out:

```
commMatrix[1:5, 1:5]
```

```
##    Ana_cla Ant_arv Ant_cor Bel_tri  Bis_pel
## 1     0.00     0.0    0.00       0 0.000000
## 5     0.00     0.0    0.00       0 0.000000
## 10    0.00     0.0    0.00       0 0.280303
## 14    0.17     0.0    0.06       0 0.170000
## 18    0.00     0.1    0.00       0 0.000000
```

```
rowSums(commMatrix)
```

```
##  1  5 10 14 18 23 27 31 36 40
##  1  1  1  1  1  1  1  1  1  1
```

Note that most functions to estimate functional diversity indices transform whatever estimate of abundance that we give into relative abundances internally. This means that in general we could use other indicators of abundance instead of relative abundances.

6.2 Required packages and additional functions

We need to load some packages in R to be able to run the different analyses we explain below. As always, these packages should be already installed at your computer before you are able to load them in R.

```
library(nlme)
library(ape)
library(TPD)
library(ks)
```

In addition, there are a series of functions that we will need to load to perform specific tasks. We will describe what each of these functions do in due time, but for now, let's simply load them:

```
source(here::here("data","chapter6","trait-flex.R"))
source(here::here("data","chapter6","wITV.R"))
```

6.3 Intra- vs interspecific trait variability within communities

Let's borrow the example from Chapter 6 in the book. Imagine a community with three species. We have measured body size in four individuals of each species:

```
spA <- c(4, 5, 6, 7)
spB <- c(6, 7, 8, 9)
spC <- c(1, 2, 5, 6)
```

We want to quantify the relative importance of ITV vs interspecific trait variability within the community. For this, we can simply decompose the sum of squares, as done in ANOVA. Lets create a matrix to make this kind of analysis, with a row for each individual measured and two columns, one containing the species to which each individual belongs and the second with the measurement of body size:

```
dataExample <- data.frame(species = rep(c("spA", "spB", "spC"), each = 4),
                          bodySize = c(spA, spB, spC))
dataExample
```

```
##    species bodySize
## 1      spA        4
## 2      spA        5
## 3      spA        6
## 4      spA        7
## 5      spB        6
## 6      spB        7
## 7      spB        8
## 8      spB        9
## 9      spC        1
## 10     spC        2
## 11     spC        5
## 12     spC        6
```

Let's make an analysis of variance of body size to see how much variability occurs within species and how much across (which will be shown by the residuals):

```
partition <- aov(bodySize~species, data = dataExample)
summary(partition)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## species      2     32      16   5.333 0.0297 *
## Residuals    9     27       3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As you can see, ITV accounts for less than half (specifically, 46%; `Sum Sq = 27`) of the total variability (`32 + 27 = 59`). Note also that the analysis reveals that, although ITV encompasses most of the trait variability in this community, there are still significant differences in body size between species (`p = 0.0297`).

Now we want to do the same type of analysis with our dataset (`traits`). In this case, we would need to repeat the ANOVA analysis for each of the plots across the environmental gradient where we have measured traits. However, the dataset has an added complication, which is that not all species have equal abundance within the plot (see `commMatrix`). Therefore, for each plot i, we want to estimate the abundance-weighted inter- and intra-specific contributions to total variance. When taken together, these two components add up to the total within-plot variance of the considered trait (in this case we will only consider height) (de Bello et al., 2011). The relative contribution of ITV to within-plot variance in height ($wITV$) can be calculated as the ratio of intraspecific variance over the total within-plot variance (Siefert et al., 2015, Fig. 6.6a in the reference book):

$$wITV_i = \frac{\sum_j \left(p_{ij} \frac{1}{N_{ij}} \sum_k (t_{ijk} - t_{ij})^2\right)}{\sum_j p_{ij}(t_{ij} - \sum_j p_{ij} t_{ij})^2 + \sum_j \left(p_{ij} \frac{1}{N_{ij}} \sum_k (t_{ijk} - t_{ij})^2\right)},$$

where p_{ij} is the relative abundance of species j in plot i, N_{ij} is the number of individuals of species j measured in the plot, t_{ijk} is the trait value of individual k of species j, and t_{ij} is the mean value of species j in the plot. Sounds complicated, and the formula can look a little intimidating. This is why we have created the function `wITV` to make these calculations. Let's start with the first plot. We need to extract the traits and species identities of all the individuals measured in plot 1, together with the relative abundance of each species:

```
# 1. traits and species indentities
plot1Data <- subset(traits, traits$Plot == 1)
# 2. relative abundances
relAbund <- commMatrix["1", as.character(unique(plot1Data$Species))]
# 3. Apply function
wITVPlot1 <- wITV(spIDs = plot1Data$Species, traitVals = plot1Data$Height, relAbund = relAbund)
wITVPlot1
```

```
## [1] 0.1540042
```

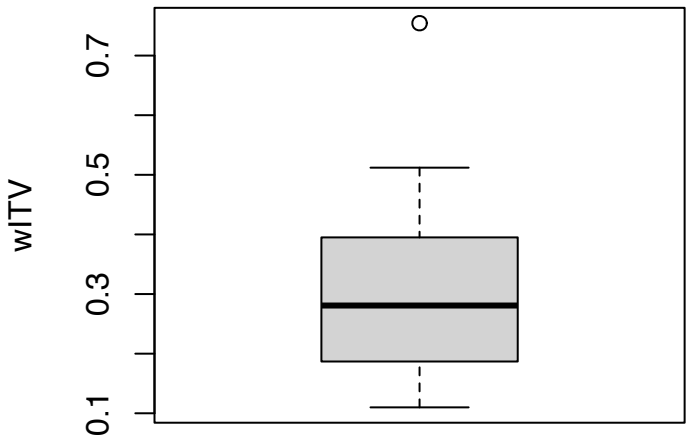
the results show that ITV accounts for 15.4% of the total trait variability in Plot 1. Note that, by incorporating species abundances we are giving more weight to the most abundant species, which means that we will get more "realistic" values of ITV. Nevertheless, if we had done the ANOVA approach as before (without considering species abundances), we would have gotten a very similar result (we encourage readers to try on their own).

After this example, we will apply the `wITV` function to all our plots, and see what is the proportion of ITV within the plots across the gradient. We will make a simple loop to do this, where we basically repeat the steps done for Plot 1. To reduce the effect of individuals with very large values we take the logarithm of plant height (see Chapter 6 in the reference book for further explanations):

```
wITVResults<-numeric()
for(i in 1:length(unique(traits$Plot))){
  commAux<-subset(traits, traits$Plot==unique(traits$Plot)[i])
  commAux$Species<-droplevels(commAux$Species)
  spNames <- unique(commAux$Species)
  relAbund<- commMatrix[i ,as.character(spNames) ]
  traitsVector <- log(commAux$Height)
  spVector <- commAux$Species
  wITVResults[i] <- wITV(spIDs = spVector, traitVals = traitsVector, relAbund = relAbund)
}
```

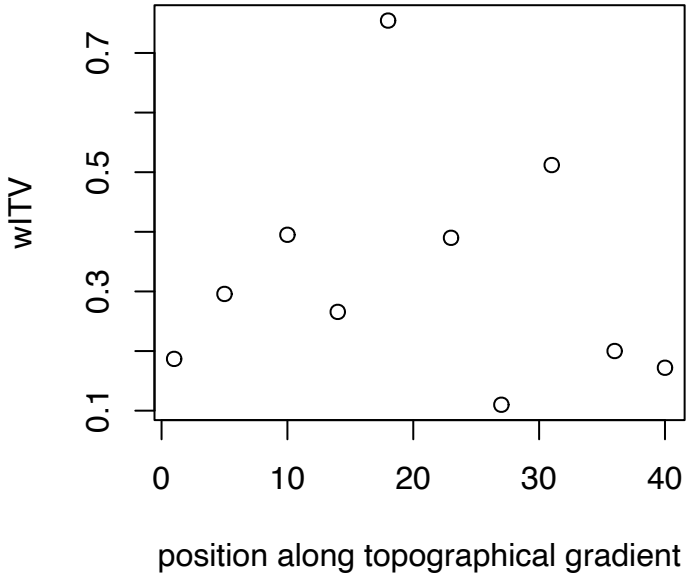
Within-plot ITV in height ranges from 10.99% and 75.43%. Let's inspect the results graphically by means of a boxplot:

```
boxplot(wITVResults, xlab = "", ylab = "wITV")
```



There is a lot of variation in plant height among our plots! At this point, it would be interesting to see if there is some pattern along the productivity gradient (remember, in this dataset, plots with higher numbers are more towards the bottom of the slope and are more productive). We can inspect this by simply plotting the relationship between wITV and the position in the gradient:

```
plot(wITVResults ~ unique(traits$Plot), xlab = "position along topographical gradient", ylab
```



It seems as if wITV in this case is not very much affected by productivity. This asks for a more thorough exploration using the environmental variables measured in each plot, but we haven't provided those.

6.4 ITV between communities

We can also estimate the contribution of ITV to the trait variation between communities in the different plots (*bITV*). The rationale behind these calculations is based on Lepš et al. (2011) and explained in more detail in Chapter 6 in the reference book. We need to estimate two different *community weighted mean (CWM)*) values (you might want to go the practical about CWM in R material Ch 5 to refresh your memory) for each plot: one considering the overall average trait value of the species (i.e. the average of the species considering all plots; CWM_{fixed}), and one considering the *local* average trait value for the species (i.e. the average of the individuals present in the plot; $CWM_{specific}$). Once

these are calculated, we can estimate the effect of intraspecific variability as $CWM_{ITV} = CWM_{specific} - CWM_{fixed}$. If we do this across all plots, we have a short of "repeated measures" of a plot, which we can analyse with models similar to a *repeated measures ANOVA*. This way, we can estimate the sum of squares associated to each of these three components: SS_{Tot}, SS_{ITV} and SS_{Fixed}, which reflect, respectively, the total variation of traits between communities, the variation due exclusively to ITV, and the variation due to species turnover (changes in species identity and abundance).

Sometimes the ITV and species turnover effects oppose each other. The *covariation* between the effects of intraspecific variability and species turnover can be calculated as:

$cov = 100 \left(\frac{SS_{Tot} - SS_{ITV} - SSFixed}{SS_{Tot}} \right)$

so, we can express SS_{Tot} as:

$SS_{Tot} = SS_{ITV} + SS_{Fixed} + cov$

Let's estimate all these elements for plant height in our dataset. For each plot, we need to compute the two CWM values (CWM_{fixed} and $CWM_{specific}$), using the script `trait-flex.R`, which we have loaded already. We will start by using `trait.transform`, a function we have developed specifically to create the matrices of traits that we need to use in each CWM calculation. For CWM_{fixed} we make a matrix in which each species has a single trait value (the grand mean) that is used across all plots, whereas for $CWM_{specific}$ we need a matrix in which each species has a different trait values in all plots in which it is present (the "local" mean). `trait.transform` requires three vectors: the sample (in these case our 10 plots, contained in `traits$Plot`), species identity (contained in `traits$Species`), and trait values measured for all our individuals (which can be found in `traits$Height`, and which we will log-transform again).

```
spID <- traits$Species
plotID <- traits$Plot
traitVector <- log(traits$Height)
traitsForCWM <- trait.transform(spIDs=as.factor(spID), traitVals=traitVector, sampleID=plotID)
```

`traitsForCWM` consists of two *plots x species* matrices: one containing the local average trait values `traitsForCWM$matLocal`, and the other containing the global average trait value of each species across all plots (so that all elements of any given column are the same: `traitsForCWM$matFixed`).

```
traitsForCWM$matLocal[, 1:5]
```

```
##     Dac_glo  Pla_lag  Tol_bar   Tri_che  Ech_pla
## 1  2.485726 1.316917 1.631683 1.2206933       NA
## 5        NA 1.114616 1.781829 0.9219977 2.377058
## 10       NA 1.514671 2.073175        NA       NA
## 14       NA 1.455400       NA 1.3994053 2.568750
## 18 3.380268 1.731947       NA 2.0820618 3.071731
## 23       NA 2.854149       NA 2.6476021       NA
## 27       NA 2.340755       NA        NA 3.479793
## 31       NA 2.563405       NA 1.8842082       NA
## 36       NA 2.185113       NA 2.4892273       NA
## 40       NA 2.186283       NA 2.1966514 3.329302
```

```
traitsForCWM$matFixed[, 1:5]
```

```
##     Dac_glo  Pla_lag  Tol_bar  Tri_che  Ech_pla
## 1  2.932997 1.926326 1.828896 1.855231 2.965327
## 5  2.932997 1.926326 1.828896 1.855231 2.965327
## 10 2.932997 1.926326 1.828896 1.855231 2.965327
```

```
## 14 2.932997 1.926326 1.828896 1.855231 2.965327
## 18 2.932997 1.926326 1.828896 1.855231 2.965327
## 23 2.932997 1.926326 1.828896 1.855231 2.965327
## 27 2.932997 1.926326 1.828896 1.855231 2.965327
## 31 2.932997 1.926326 1.828896 1.855231 2.965327
## 36 2.932997 1.926326 1.828896 1.855231 2.965327
## 40 2.932997 1.926326 1.828896 1.855231 2.965327
```

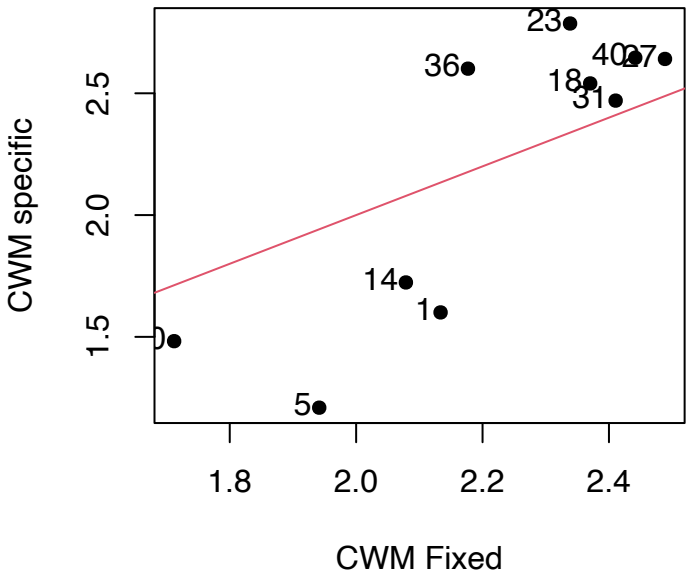
These trait values, along with the matrix containing the abundance of each species in each plot, to estimate the two CWM (fixed and specific) for each plot, with the function `trait.CWM` (which is also contained in the `trait-flex.R` file):

```
CWMSpecFix <- trait.CWM(matLocal=traitsForCWM$matLocal, matFixed=traitsForCWM$matFixed,
                        commMatrix=commMatrix)
CWMSpecFix
```

```
## $specific.avg
##        1        5       10       14       18       23       27       31
## 1.600343 1.209261 1.482333 1.723393 2.540519 2.786522 2.641063 2.469973
##       36       40
## 2.601152 2.646004
##
## $fixed.avg
##        1        5       10       14       18       23       27       31
## 2.133575 1.941581 1.712001 2.078686 2.370321 2.338474 2.488667 2.410578
##       36       40
## 2.176895 2.441649
```

It is possible to get these pairs of CWM values without using the `trait.transform` and `trait.CWM` functions (for example, writing your own code by hand). Regardless of how this is achieved, now we have our pairs of CWM values estimated. Let's take a look at them (we will add a label to each point to see to which plot each point belongs; remember that plots with higher values should have higher productivity):

```
plot(CWMSpecFix$specific.avg ~ CWMSpecFix$fixed.avg, pch = 16,
     xlab = "CWM Fixed", ylab = "CWM specific")
text(x = CWMSpecFix$fixed.avg, y = CWMSpecFix$specific.avg,
     labels = names(CWMSpecFix$specific.avg), pos = 2, offset = 0.2)
abline(0, 1, col = 2)
```



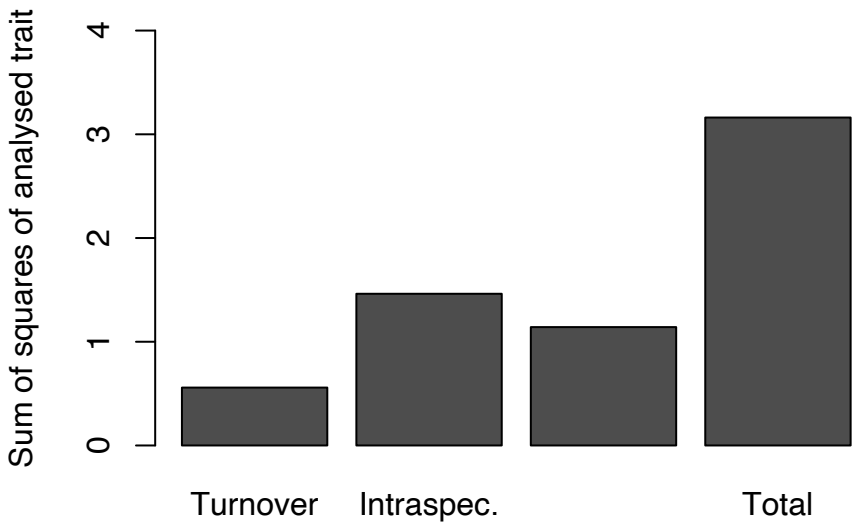
We have to take into account that *specific.avg* reflects the CWM value observed in each plot, whereas *fixed.avg* shows what this value would be if we used a single trait value for each species obtained by considering all plots. As you can see in the plot, CWM_{fixed} is higher than $CWM_{specific}$ in the plots with smaller average plant height (those with lower productivity), and this situation reverses in the plots with larger height values. This means that using global average trait values overpredicts local height in less productive conditions and underpredicts plant height under more productive conditions, and indicates that the covariation between the two components will be positive. In other words, in the productive plots there are species that are taller, but the individuals of species that occur also in low productivity conditions are taller as well, and vice versa in the less productive plots. To examine that phenomenon, we will use the function `trait.flex.anova`, first performing an intercept-only linear model:

```
bITV <- trait.flex.anova(formula = ~1, specif.avg = CWMSpecFix$specific.avg,
                         fixed.avg = CWMSpecFix$fixed.avg)
bITV
```

```
##
##  Decomposing trait sum of squares into composition turnover
##  effect, intraspecific trait variability, and their covariation:
##       Turnover Intraspec. Covariation  Total
## Total  0.55798     1.4625      1.1412 3.1616
##
##  Relative contributions:
##       Turnover Intraspec. Covariation Total
## Total   0.1765     0.4626      0.3609     1
```

`bITV` contains first the sum of squares corresponding to each of the components of trait variation across plots: turnover (changes in species across plots), intraspecific (within-species trait adjustments) and their covariation. The second line (relative contribution) simply shows the proportional contribution of each of these components (note that "Total" adds up to 1 in that case). We can visually explore these values too:

```
plot(bITV)
```



What lesson can we learn from this exercise? It seems that in a short spatial gradient with limited species turnover, ITV is more importance than species turnover. Further, covariation contributes significantly to the total variability of plant height. This means that in plots where taller species are more abundant, individuals of a particular species are also taller than their average height across the gradient, so that the two effects reinforce each other.

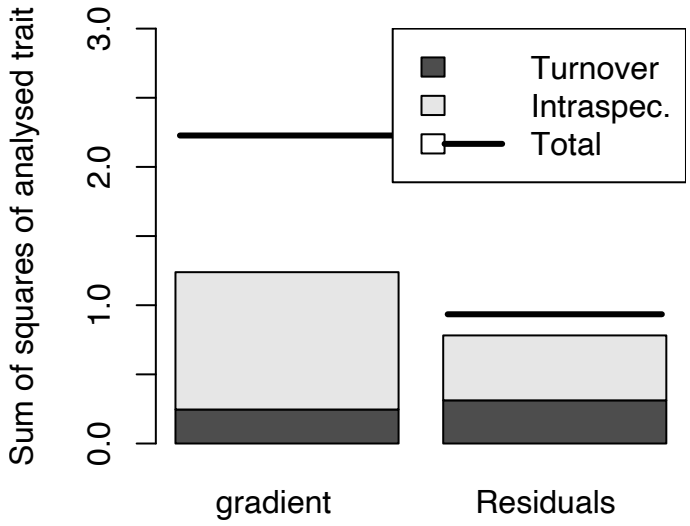
We can refine the analyses and add environmental explanatory variables to understand the observed patterns, in this case the position (i.e. productivity) along the gradient:

```
gradient <- as.numeric(names(CWMSpecFix$specific.avg))
bITV2 <- trait.flex.anova(formula = ~gradient, specif.avg = CWMSpecFix$specific.avg,
                          fixed.avg = CWMSpecFix$fixed.avg)
bITV2
```

```
##
##  Decomposing trait sum of squares into composition turnover
##  effect, intraspecific trait variability, and their covariation:
##             Turnover Intraspec. Covariation  Total
## gradient     0.24587    0.99301     0.98823 2.2271
## Residuals    0.31211    0.46946     0.15292 0.9345
## Total        0.55798    1.46248     1.14115 3.1616
##
##  Relative contributions:
##             Turnover Intraspec. Covariation  Total
## gradient     0.07777     0.3141     0.31257 0.7044
## Residuals    0.09872     0.1485     0.04837 0.2956
## Total        0.17649     0.4626     0.36094 1.0000
##
##  Significance of testable effects:
##             Turnover Intraspec.     Total
## gradient     0.03635  0.0033739 0.0023919
```

The table of results is a bit more complex now, although the summary is basically the same as in a repeated measures ANOVA. A lot of the functional variation in our dataset is associated with the position of the plot across the productivity gradient (70.4%). This is particularly important for ITV, although the effect of the position across the gradient is also significant for the part of trait variation due to species turnover. The rest of the variation takes place within plots (contained in "Residuals"). Again, we can plot these results:

```
plot(bITV2)
```



The dark grey part of the columns corresponds to species turnover, whereas the light grey part corresponds to ITV. As for the numerical results, the variability explained by the position across the gradient is mostly attributed to ITV. The black line above the columns indicate total variation between ITV and species turnover. When the bar is positioned

above the column, the covariation is positive. There are cases in which this covariation can be negative. This occurs if turnover and ITV "compensate" each other. This would correspond to a case in which taller species in a plot are selected for, smaller individuals would be promoted (see Lepš et al., 2011, for an example with other traits).

6.5 Decomposition of variance across scales using mixed models

So far we have decomposed trait variability into a within and between species component, and explored how explanatory variables (in our case, the position along the gradient) affect this decomposition. There are cases in which we might want to decompose trait variability across more levels. Messier et al. (2010) propose an alternative way to decompose variation in plant traits across ecological scales that can be useful for this task. They use a method to decompose variability in leaf traits across six nested ecological scales (i.e. site, plots within sites, species within plots, individual trees within species, canopy strata within individual trees and different leaves within each canopy strata). Carmona et al. (2015b) use a similar approach applied to the same dataset we are using in these examples. Basically, one has to assign the measured trait values to the appropriate scales that are considered. In our example, variation in traits in the dataset is due to differences 1) among individuals of the same species within plots 2) among species within plots and 3) among plots To decompose total trait variability into these three scales we can fit a mixed effects model using the nested scales (`Species` within `Plots`) as a random factor. We can do this using the `lme` function from the package `nlme`, followed by the function `varcomp` from `ape`. Again, we will consider plant height (after log-transformation):

```
logHeight <- log(traits$Height)
modPart <- lme(logHeight ~ 1, random = ~ 1 | Plot / Species, data = traits, na.action = na.omit)
varcompHeight <- ape::varcomp(modPart, scale = 1)
varcompHeight
```

```
##      Plot   Species    Within
## 0.3578957 0.5118311 0.1302731
## attr(,"class")
## [1] "varcomp"
```

The logic behind this method is a bit different than in the two previous analyses, since it encompasses both the quadrat scale and the gradient scale. Height differences between the individuals of the same species in the same quadrat account for around 13% of the variability. On the other hand, differences among species within quadrats account for ca. 51% of the total, whereas the differences between quadrats accounts for around 36%. We can even get confidence intervals for these proportions by means of a bootstrap resampling. The bootstrap consists in sampling with replacement as many individuals as we have in the dataset. Then, some individuals will be present more than once in the *bootstrap sample*, whereas some individuals from our dataset will be absent from it. Confidence intervals for the proportions can be then achieved by doing the bootstrap procedure many times (here 1,000 times) and estimating the decomposition of variance each time.

```
nreps <- 1000
varPlot <- varSpecies <- varWithin <- numeric()
for (i in 1:nreps){
  data.aux <- traits[sample(1:nrow(traits), nrow(traits), replace = T), ]
  logHeightAux <- log(data.aux$Height)
  varcomp.aux <- ape::varcomp(lme(logHeightAux ~1 , random = ~1 | Plot / Species,
                              data = data.aux, na.action = na.omit), scale = 1)
  varPlot[i] <- varcomp.aux["Plot"]
```

```
  varSpecies[i] <- varcomp.aux["Species"]
  varWithin[i] <- varcomp.aux["Within"]
}
```

We now have the confidence intervals for each scale. For example, 51.18% of the variability in plant height in our dataset is associated with differences among species within plots We can now say that

```
quantile(varSpecies, probs=c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.4881252 0.5617823
```

the confidence interval for this percentage is (48.81% – 56.18%).

6.6 The Trait Probability Density (TPD) approach

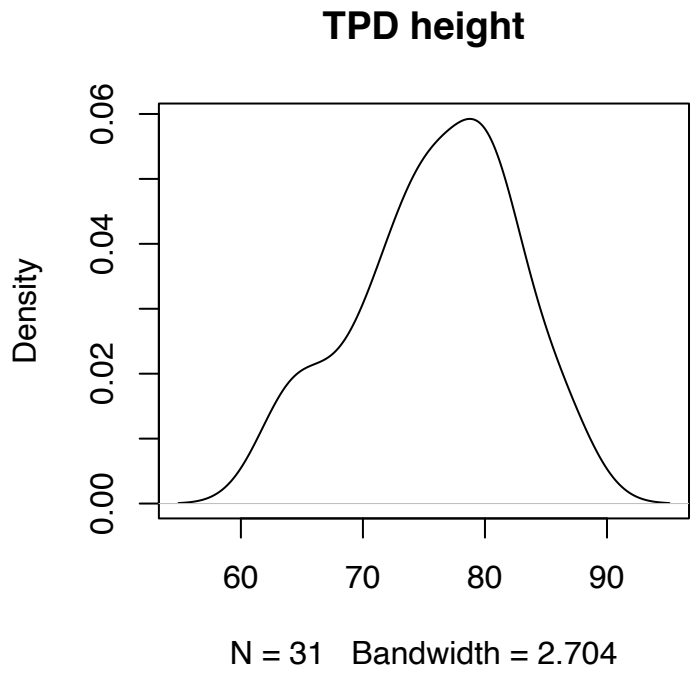
6.6.1 The probabilistic nature of functional diversity

The species that make a community display a variety of trait values. As early as 1957, Hutchinson proposed his concept of niche, which is the multidimensional hypervolume, in which each dimension represents an essential resource, and in which a species can maintain a population. Hutchinsonian hypervolumes are generally conceived as uniform, and all the points contained within them are assigned an equal probability of persistence of the species. Consequently, these hypervolumes can be characterized by their boundaries.

However, this is not a totally satisfying conception, because some parts of the species niche have better conditions than others for a particular individual (and its trait values). Therefore, niches of species can be described as probability density functions. The same idea is applicable to the trait distributions of individuals within a species, as individuals vary in their trait values. This means that some trait values are more likely than others, just as it is more likely to find a 1.70 m tall person than a 2.10 m tall one. Moreover, this also holds for combinations of trait values, as many traits are not completely independent from others (it is unlikely to find a 2.10 m tall person, but even more so that this person weights 60 kg). Probability density functions can be used to reflect the unequal probabilities of different trait values. We will refer to these functions as Trait Probability Density (TPD).

Consider for instance the dataset `trees`, contained in the package `datasets` which is part of the base distribution of R. It contains 31 observations of the girth, height and volume of black cherry trees. Let us calculate the TPD of the trait `Height` for this population of cherry trees:

```
kde_height <- density(trees$Height)
plot(kde_height, main = "TPD height")
```



The plot shows clearly that heights around 75 ft are much more likely than heights higher than 90 ft.

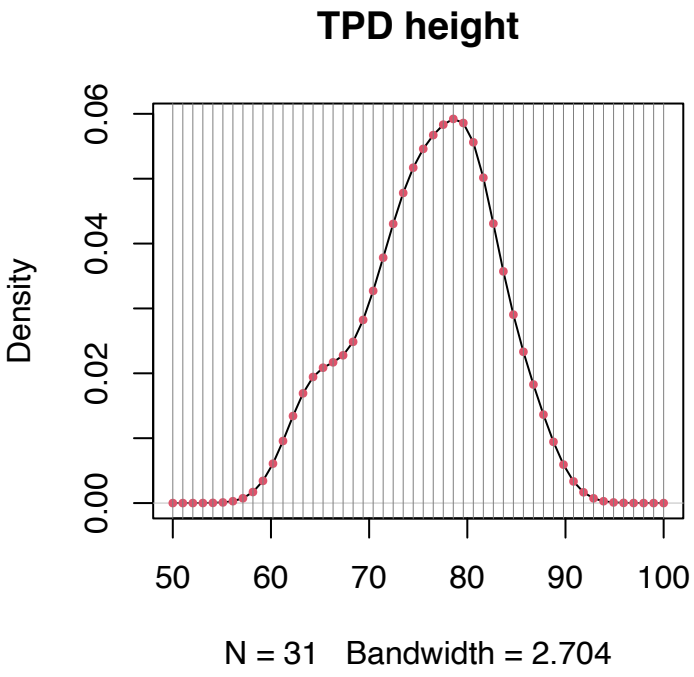
TPD's are probability density functions, therefore they integrate to 1, which means that the area under the plotted curve equals 1. To calculate this integral, we will first expand the range of traits of the database –we consider values between 50 and 100 ft; we want to be sure that the interval we choose encompasses all the distribution. Then, we divide this range into equal-length intervals. Although it is best to use a high number of intervals, in this example, for the sake of visibility, we will divide it into 50 intervals:

```
grid <- seq(from = 50, to = 100, length.out = 50)
edge_length <- grid[2] - grid[1]
#Each interval has this size (in trait units, ft in this case):
edge_length
```

```
## [1] 1.020408
```

Now, let's evaluate the TPD function in each point of the grid.

```
kde_height <- density(trees$Height, from = 50, to = 100, n = 50)
plot(kde_height, main = "TPD height")
points(kde_height$x, kde_height$y, pch = 16, col = 2 , cex=0.6)
abline(v = grid, col = "grey50", lwd = 0.3)
```



The red points represent the value that the TPD function takes in each of the 50 intervals. We can transform each of the intervals into a rectangle with the base given by `edge_length` = 1.02 and the height by the value of the TPD function evaluated in each point (contained in `kde_height$y`). If we then sum the areas of these rectangles, we find that:

```
sum(edge_length * kde_height$y)
```

```
## [1] 1.000979
```

This sum is roughly 1, due to the fact that TPD functions reflect the probability of observing some specific value for a given variable (in this case a specific height), and the sum of all possible values must be 1. The `TPD` package allows to compute TPD functions at different ecological scales. The package allows estimating TPDs for single or multiple traits.

Now, we will show some of the basic features of the `TPD` package. More advanced and detailed examples can be found in the package vignette `vignette("TPD")`. The most basic function in the package is the `TPDs` function, which basically allows estimating the TPD of individual species.

6.6.2 The `TPDs` function

To illustrate the use of `TPDs`, we will again consider the `traits` data we have been using so far. Suppose that we want to calculate the TPDs of each species considering `height` and `SLA`. Estimating these TPDs's is very simple:

```
traits_matrix <- traits[, c("Height", "SLA")]
sp_matrix <- traits$Species
TPDs_result <- TPDs(species = sp_matrix, traits_matrix)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

After a few moments of calculations, the object `TPDs_result` is created. This object contains, along with other information, the grid of cells (combinations of Height and SLA values) in which the function was evaluated, contained in `TPDs_result$data$evaluation_grid`:

```
head(TPDs_result$data$evaluation_grid)
```

```
##      Height       SLA
## 1 -6.200000 -1.860134
## 2 -5.886432 -1.860134
## 3 -5.572864 -1.860134
## 4 -5.259296 -1.860134
## 5 -4.945729 -1.860134
## 6 -4.632161 -1.860134
```

```
tail(TPDs_result$data$evaluation_grid)
```

```
##         Height      SLA
## 39995 54.63216 73.93828
## 39996 54.94573 73.93828
## 39997 55.25930 73.93828
## 39998 55.57286 73.93828
## 39999 55.88643 73.93828
## 40000 56.20000 73.93828
```

There are a lot of cells in the grid, resulting from dividing each axis into 200 equal intervals (which is the function's default number for two dimensions).

The object `TPDs_result$TPDs` is a list with one element for each species:

```
names(TPDs_result$TPDs)
```

```
##  [1] "Dac_glo" "Pla_lag" "Tol_bar" "Tri_che" "Ech_pla" "Leo_tar" "Bis_pel"
##  [8] "Cha_mix" "Pet_nan" "Pla_cor" "Spe_pur" "Tri_tom" "Ana_cla" "Ant_cor"
## [15] "Tri_pan" "Ant_arv" "Psi_inc" "Vul_cil" "Cre_tar" "Ery_cam" "Tri_glo"
## [22] "Bro_hor" "Bro_rub" "Hor_mur" "Sis_cav" "Tri_str" "Med_sat" "Nea_apu"
## [29] "Orn_com" "Bel_tri"
```

Each element of this list contains the probability associated to each cell for that species (basically, the TPD function for each species). As we saw before, the sum of all TPD values across all cells for each is 1:

```
sapply(TPDs_result$TPDs, sum)
```

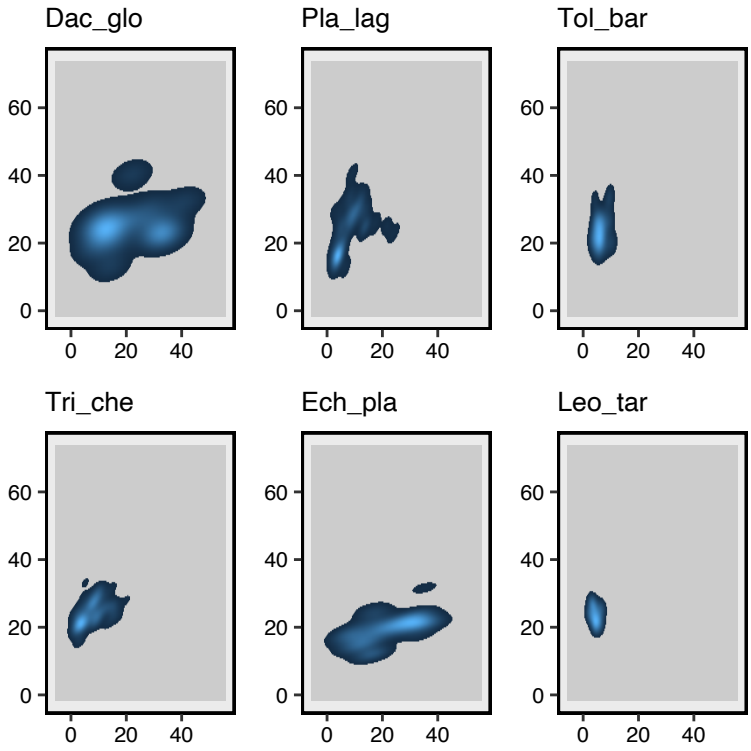
```
## Dac_glo Pla_lag Tol_bar Tri_che Ech_pla Leo_tar Bis_pel Cha_mix Pet_nan Pla_cor
##       1       1       1       1       1       1       1       1       1       1
## Spe_pur Tri_tom Ana_cla Ant_cor Tri_pan Ant_arv Psi_inc Vul_cil Cre_tar Ery_cam
##       1       1       1       1       1       1       1       1       1       1
## Tri_glo Bro_hor Bro_rub Hor_mur Sis_cav Tri_str Med_sat Nea_apu Orn_com Bel_tri
##       1       1       1       1       1       1       1       1       1       1
```

The function `plotTPD` can be used to visualize the resulting TPDs's. Note that it can only handle 1 or 2-dimensional data. Since we have 30 species, and it takes a while to plot them all at once, we can select a few species using the argument `whichPlot`:

```
plotTPD(TPD = TPDs_result, whichPlot = c(1:6), nRowCol = c(2,3))
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



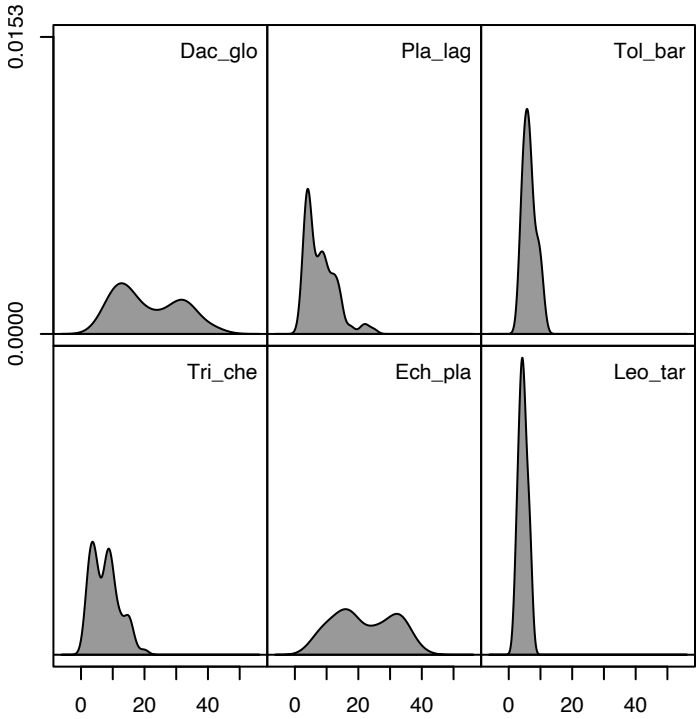
Each plot shows the TPD of one of the selected six species, which reflects the probability of observing certain combinations of trait values (height is in the x axis; SLA is in the y axis). TPD functions are somehow like a landscape, in which peaks correspond to trait combinations with higher probabilities, here shown in lighter tones of blue, whereas lower probabilities have darker tones of blue; the grey area corresponds to the part of the functional space that is not occupied by the species (probability is 0).

Now that we are able to visualize TPD's, we are going to show an important aspect of their construction: alpha. The parameter `alpha` determines the proportion of the probability density function of each species or population that will be included in the resulting TPD (Blonder et al., 2014; Swanson et al., 2015). Setting `alpha = 1` will include the whole function, whereas `alpha = 0.5` will include only the 50% of highest probability (lighter blue). It is important to note that when `alpha` < 1, the `TPD` function rescales the probabilities so that they add up to 1. By default, `alpha` is set to 0.95 to make `TPDs` not too sensitive to outliers. To visualize the effect of alpha in the calculation of a 1-dimensional TPD, let's first set `alpha = 1`:

```
traits_matrix_d1 <- traits_matrix[, c("Height")]
TPDs_result_d1_a1 <- TPDs(species = sp_matrix, traits = traits_matrix_d1, alpha=1)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDs_result_d1_a1, whichPlot = c(1:6), nRowCol = c(2,3))
```

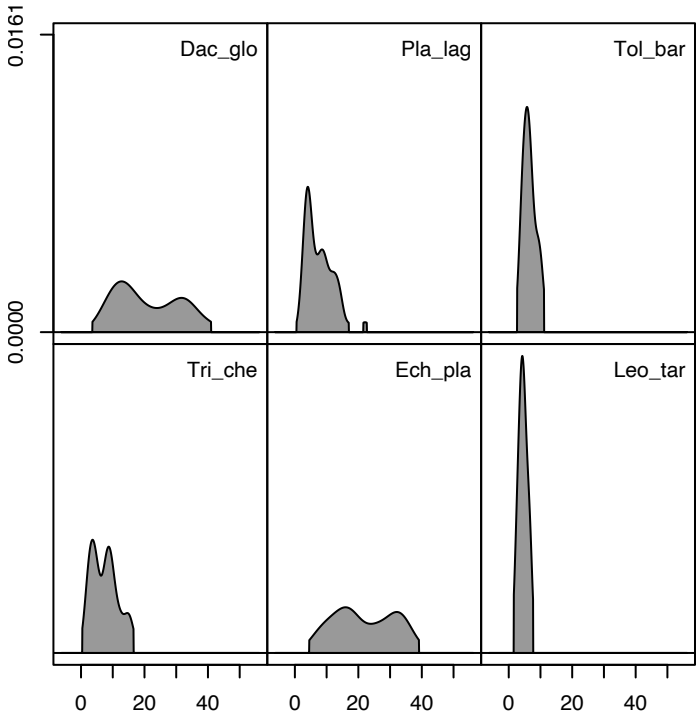


Now, `alpha = 0.95`, the default value:

```
TPDs_result_d1_a095 <- TPDs(species = sp_matrix, traits = traits_matrix_d1, alpha=0.95)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDs_result_d1_a095, whichPlot = c(1:6), nRowCol = c(2,3))
```

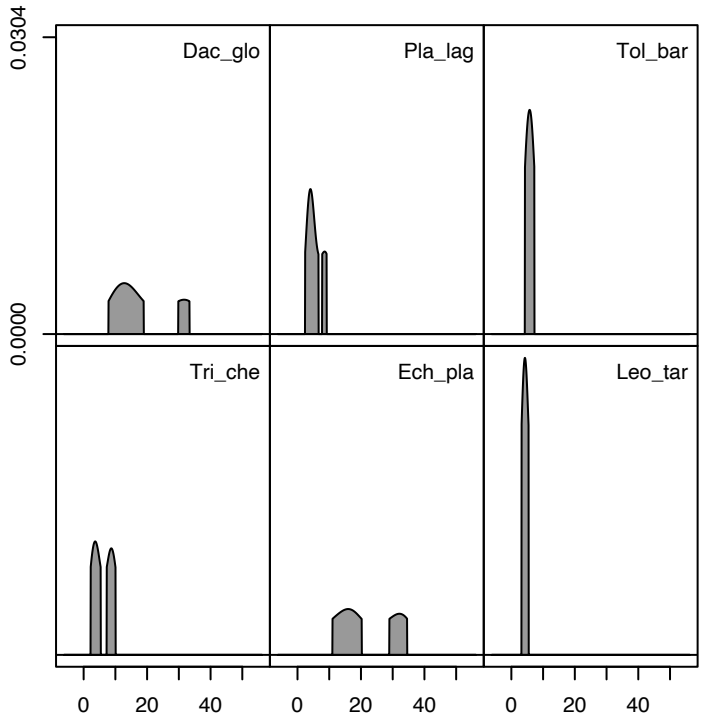


Can you see how the edges of the distribution are truncated now? this is how `alpha = 0.5` looks like:

```
TPDs_result_d1_a05 <- TPDs(species = sp_matrix, traits = traits_matrix_d1, alpha=0.5)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDs_result_d1_a05, whichPlot = c(1:6), nRowCol = c(2,3))
```



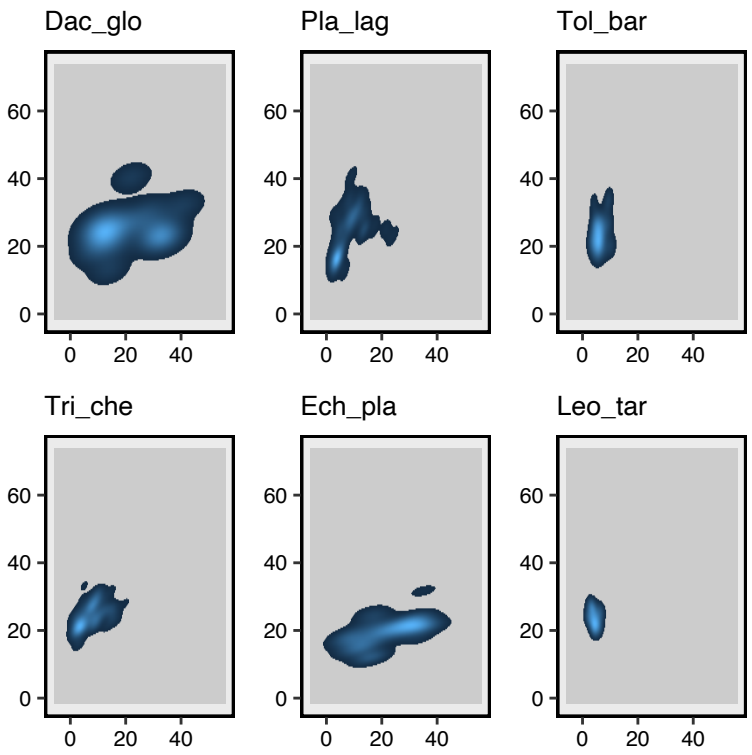
Indeed, the same logic applies for higher dimensions. Let's briefly see the two dimensional case:

```
TPDs_result_a95 <- TPDs(species = sp_matrix, traits = traits_matrix, alpha=0.95)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDs_result_a95, whichPlot = c(1:6), nRowCol = c(2,3))
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```

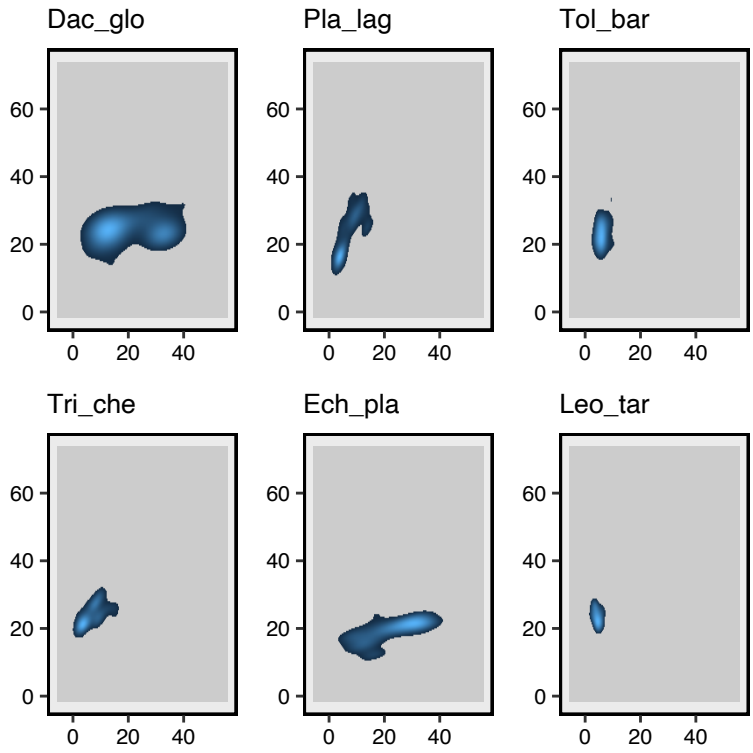



```
TPDs_result_a75 <- TPDs(species = sp_matrix, traits = traits_matrix, alpha=0.75)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDs_result_a75, whichPlot = c(1:6), nRowCol = c(2,3))
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



All these estimations are based on data of all the measured individuals of each species across all plots. But we have measured traits on individuals of each species in different plots, and we have already seen before that the position along the gradient affects trait values! The argument `samples` of `TPDs` allows us to calculate a separate TPDs for each population of each species. This information is contained in `traits$Plot`. Let's estimate it and plot the TPDs of the populations of *Plantago lagopus*, the most abundant plant in our dataset, across the productivity gradient (from less to more productive plots):

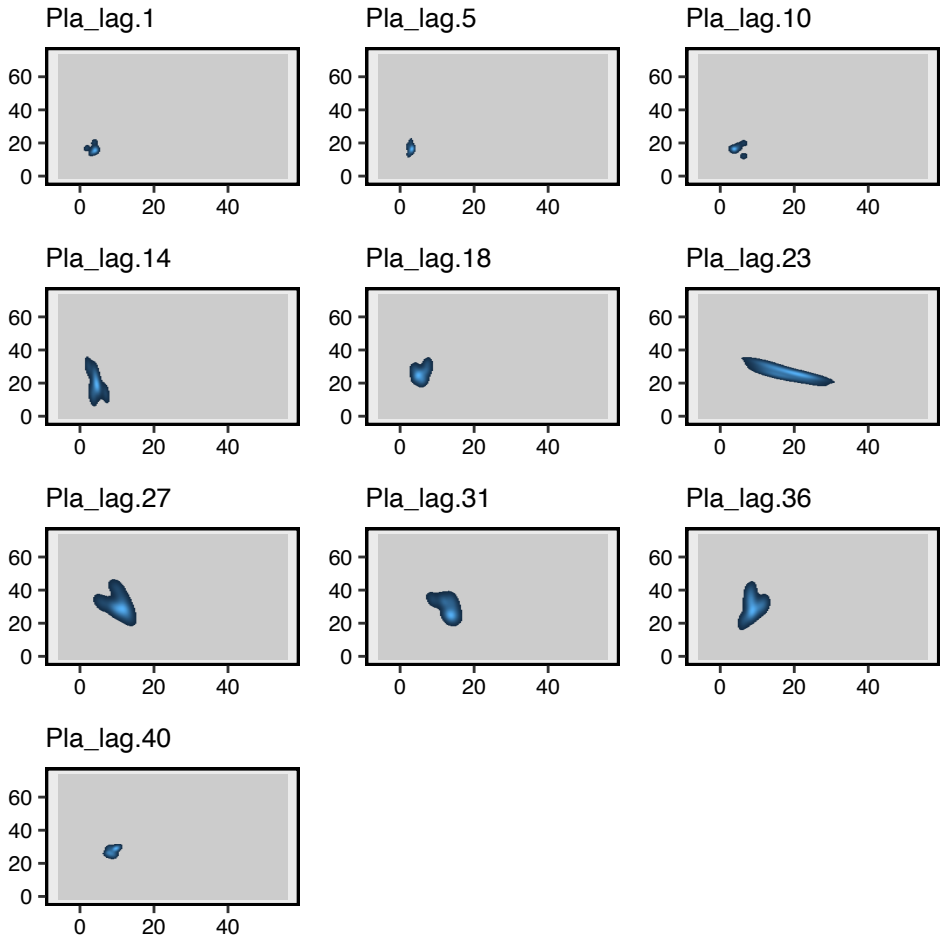
```
TPDs_pops <- TPDs (species = sp_matrix, traits = traits_matrix, samples = traits$Plot)
```

```
## -------Calculating densities for Multiple populations_Multiple species-----------
```

```
plantagoPops <- which(substr(names(TPDs_pops$TPDs),  1, 7) == "Pla_lag")
plotTPD(TPD = TPDs_pops, whichPlot = plantagoPops)
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



The environment affect the traits of our species! Very unproductive sites (those with small numbers) produce very small individuals (height is in the x axis) with relatively low SLA (y axis). Furthermore, those individuals are all very similar between them. However, as we go downslope and fertility increases, we tend to find taller individuals with higher SLA, and much higher variability among individuals.

6.6.3 The `TPDsMean` function

The `TPDs` function requires a high amount of information to work: one measurement per individual and trait considered. In addition, several individuals are required to estimate reliable TPDs's. We do not usually have access to such high-quality information. The `TPDsMean` function is designed for those cases. `TPDsMean` performs the same task as `TPDs`, but it estimates the probabilities by calculating a multivariate normal distribution for each species or population, rather than using kernel density estimations. Hence, the only information required is the mean and standard deviation of each trait considered for each species or population. Let us imagine that we only have the means and standard deviations of our species:

```
names_sp <- unique(traits$Species)
mt1 <- tapply(traits[, "Height"], traits$Species, mean)
```

```
mt2 <- tapply(traits[, "SLA"], traits$Species, mean)
means_traits <- matrix(c(mt1, mt2), ncol=2)
head(means_traits)
```

```
##         [,1]     [,2]
## [1,]  7.3200 17.50411
## [2,] 17.0925 27.25867
## [3,]  5.5000 18.72288
## [4,] 24.8700 24.95832
## [5,]  7.3200 34.82942
## [6,]  9.5000 26.80517
```

```
st1 <- tapply(traits[, "Height"], traits$Species, sd)
st2 <- tapply(traits[, "SLA"], traits$Species, sd)
sds_traits <- matrix(c(st1, st2), ncol=2)
head(sds_traits)
```

```
##          [,1]      [,2]
## [1,] 1.828053  4.649635
## [2,] 5.928349  4.201657
## [3,] 1.666667  3.848222
## [4,] 8.385974  2.729603
## [5,] 3.864826 10.783115
## [6,] 1.649916  5.054252
```

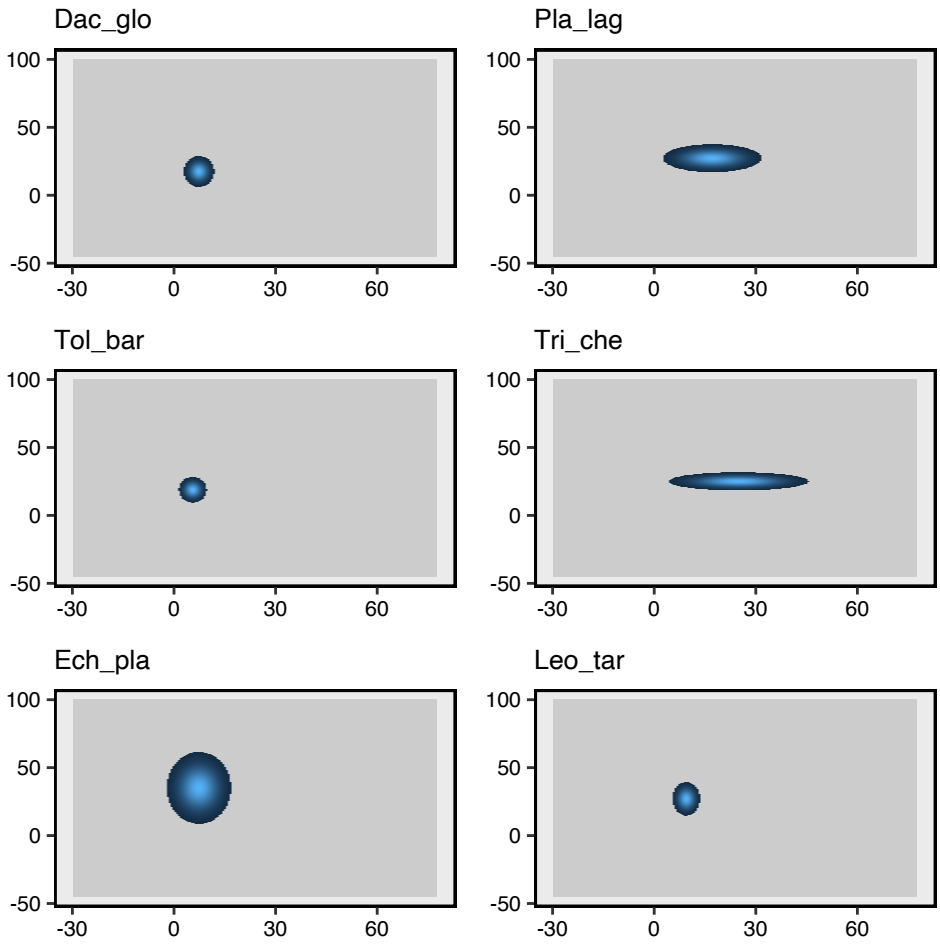
Let's build TPDs functions based on those means and standard deviations:

```
TPDsMean_results<- TPDsMean(species = names_sp, means = means_traits, sds = sds_traits)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPD = TPDsMean_results, whichPlot = c(1:6))
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



6.6.4 Estimating functional dissimilarity: the `dissim` function

Now that we have learnt how to build TPDs functions for species (or populations), we will discuss one of the most basic (and useful) features. Probability density functions were initially used in trait-based ecology to estimate the trait dissimilarity between species while considering ITV. The idea behind this is that when two species share some part of the functional niche, their respective TPDs functions must overlap up to some degree. We recommend reading Mouillot et al. (2005), Lepš et al. (2006), Geange et al. (2011) or de Bello et al. (2013), as well as Chapter 6 in the reference book, as useful references to understand this concept and the related technical, mathematical and ecological details behind the concept of overlap. To illustrate this method, we are going to use a reduced subset of species. Let's select the first six species and estimate their TPDs considering the trait `Height`:

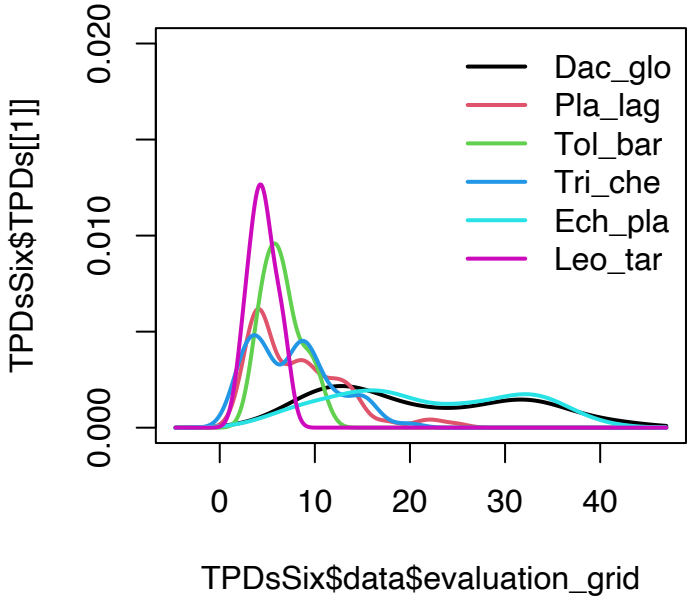
```
firstSix <- droplevels(unique(traits$Species)[1:6])
traitsSix <- subset(traits, traits$Species %in% firstSix)
TPDsSix <- TPDs(species = droplevels(traitsSix$Species), traits = traitsSix$Height, alpha = 1)
```

```
## -------Calculating densities for One population_Multiple species-----------
```

Let's plot these TPDs together:

```
par(mfrow=c(1, 1))
plot(TPDsSix$data$evaluation_grid, TPDsSix$TPDs[[1]], type="n",
  ylim = c(0, 0.02))

for (i in 1:length(TPDsSix$TPDs)){
```

```
  lines(TPDsSix$data$evaluation_grid, TPDsSix$TPDs[[i]], lwd=2, col=i)
}
legend("topright", bty = "n", lwd = 2, col = 1:length(TPDsSix$TPDs),
  legend = names(TPDsSix$TPDs)[1:6])
```



The plot shows that some species are rather similar in their height distribution, as shown by the relatively high overlap between their TPDs's (for example Ech_pla and Dac_glo). Other species are very different, with only a very small proportion of their TPDs's overlapping (e.g. Leo_tar and Ech_pla). We can use the `dissim` function to accurately estimate these differences:

```
dissimSix <- dissim(TPDsSix)
```

```
## Calculating dissimilarities between 6 populations. It might take some time
```

```
dissimSix$populations$dissimilarity
```

```
##            Dac_glo   Pla_lag   Tol_bar   Tri_che    Ech_pla   Leo_tar
## Dac_glo 0.00000000 0.5655609 0.7727921 0.5837351 0.09422965 0.8932964
## Pla_lag 0.56556088 0.0000000 0.3267787 0.1322225 0.63196992 0.4465392
## Tol_bar 0.77279213 0.3267787 0.0000000 0.3403386 0.81418699 0.3632335
## Tri_che 0.58373511 0.1322225 0.3403386 0.0000000 0.63684057 0.4944688
## Ech_pla 0.09422965 0.6319699 0.8141870 0.6368406 0.00000000 0.9076970
## Leo_tar 0.89329638 0.4465392 0.3632335 0.4944688 0.90769703 0.0000000
```

The dissimilarity matrix quantifies these differences. Indeed, as we predicted the dissimilarity between Ech_pla and Dac_glo is very low (0.09), and the dissimilarity between Leo_tar and Ech_pla is much closer to 1 (0.91), which is the maximum value that dissimilarity can take (see Chapter 6 of the reference book).

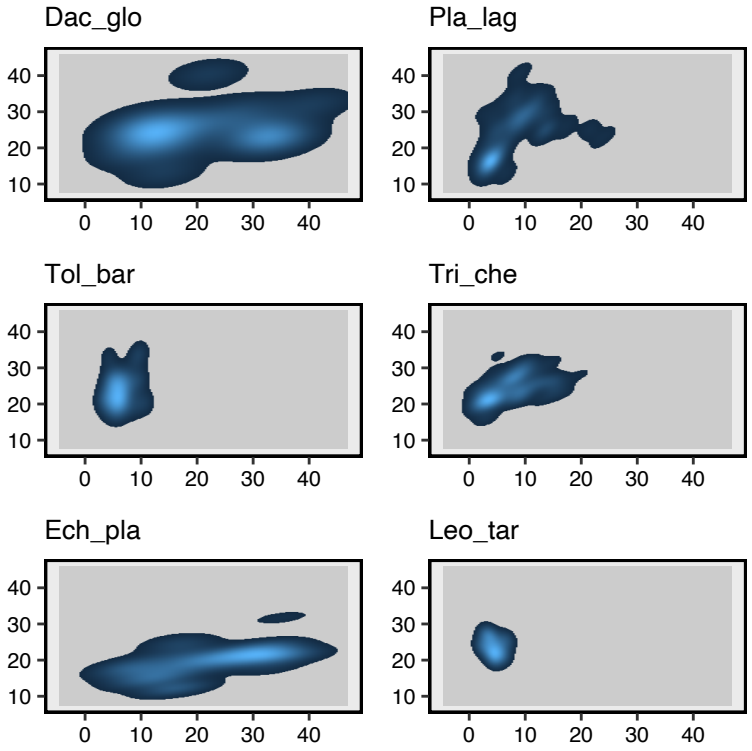
The function works similarly for higher niche dimensions. Let's estimate TPDs based on both traits for the same subset of species:

```
TPDsSix2d <- TPDs(species = droplevels(traitsSix$Species), traits = traitsSix[, c("Height",
```

```
## -------Calculating densities for One population_Multiple species-----------
```

```
plotTPD(TPDsSix2d)
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



The plot now shows even bigger dissimilarities between species (not that overlap-based dissimilarity can only increase as we consider more traits). For example, the overlap between Dac_glo and Leo_tar seems very small. Let's estimate those dissimilarities:

```
dissimSix2d <- dissim(TPDsSix2d)
```

```
## Calculating dissimilarities between 6 populations. It might take some time
```

```
dissimSix2d$populations$dissimilarity
```

```
##           Dac_glo   Pla_lag   Tol_bar   Tri_che   Ech_pla   Leo_tar
## Dac_glo 0.0000000 0.7067361 0.8024373 0.6680376 0.5544915 0.9136120
## Pla_lag 0.7067361 0.0000000 0.5278488 0.4367696 0.8647449 0.7403470
## Tol_bar 0.8024373 0.5278488 0.0000000 0.4309327 0.8983680 0.4362647
```

```
## Tri_che 0.6680376 0.4367696 0.4309327 0.0000000 0.9039688 0.5786795
## Ech_pla 0.5544915 0.8647449 0.8983680 0.9039688 0.0000000 0.9795043
## Leo_tar 0.9136120 0.7403470 0.4362647 0.5786795 0.9795043 0.0000000
```

In addition to overlap-based dissimilarity, the `dissim` function also decomposes dissimilarity into shared and non-shared parts of functional space. We will see what these consist on later, when we explore beta diversity.

6.6.5 `TPDc`: from species to communities

One interesting feature of TPD functions is that the concept can be easily scaled up to express the functional structure of communities (*TPDc*). The `TPDc` function combines the TPDs of the species with information about their abundances in communities. In other words TPDc is the result of summing the probability of each of the species present in the community, weighted by their relative abundances.

We can illustrate this concept with the `traits` example, using the plots as "communities". To calculate the TPDc function of each plot we first have to calculate the TPDs of *all* the species. (This step is important: if you do not have the TPDs of all the species or populations, you cannot estimate the TPDc of the plot, and `TPDc` will end with an error). Since we have information at the within-plot scale, we will use the `TPDs_pops` object that we created above:

```
TPDc_result <- TPDc(TPDs = TPDs_pops, sampUnit = commMatrix )
```

What is new in TPDc objects compared to TPDs objects is the information contained in its component `$TPDc`. Specifically, we are interested in the element `TPDc_result$TPDc$TPDc`, which is a list with an element for each plot (`length(TPDc_result$TPDc$TPDc) =` 10), each one containing the probability associated to each cell of the grid in the corresponding plot. In fact, TPDc reflects the probability of observing a given trait value when extracting a random individual from the community. Because TPDc's are probability density functions, they still integrate to 1:

```
sapply(TPDc_result$TPDc$TPDc, sum)
```

```
##  1  5 10 14 18 23 27 31 36 40
##  1  1  1  1  1  1  1  1  1  1
```

(NOTE: as with TPDs functions, the probabilities contained in a TPDc will always sum 1, because relative abundances are used to weight TPD's)

Similar to what we did with TPDs's, `plotTPD` can be used to provide a graphical representation of TPDc's based on 1 or 2 dimensions:

```
plotTPD(TPD = TPDc_result)
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'
```

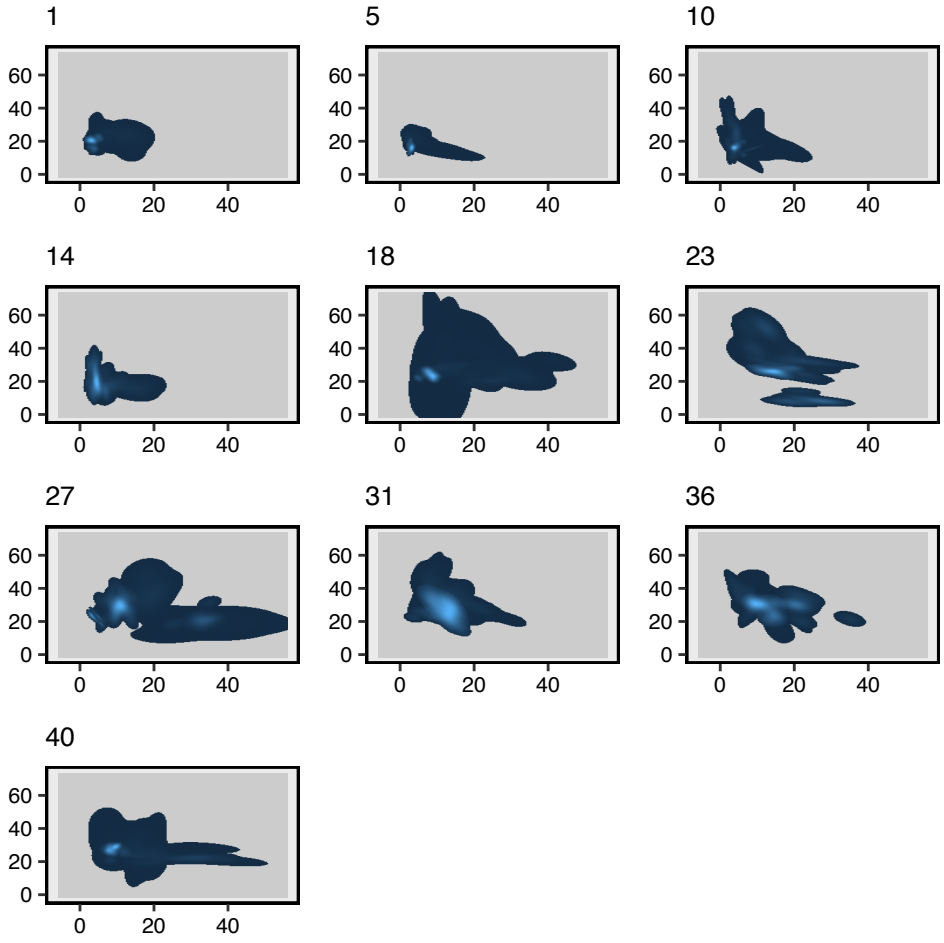
```
## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



The plots show how the functional structure of the plant communities (the plots) change along the productivity gradient (remember, higher values are associated to higher productivity). Similarly to the *Plantago lagopus* example, communities occupy a smaller proportion of the trait space in the less productive sites. However, the functional space they occupy also decreases again in the lowest end of the gradient. This pattern could be due to many things: are communities in the upper part less species-rich? or simply the species are more redundant and there is more overlap between them? Is the functional space evenly occupied? As we saw in Chapter 5 in the reference book, many indices try to quantify different aspects of the trait structure of communities and tackle these questions. The TPD package includes estimations of some of these indices, and we will dedicate the rest of this document to briefly explore them.

6.6.6 Functional Richness, Evenness and Divergence (`REND` function)

These three indices of functional diversity were originally described by Mason et al. (2005). We have seen them in the Chapter 5 of the reference book, and described how to calculate them using the `dbFD` package in the section showing how to estimate functional diversity indices. In the `TPD` package these indices can be estimated at whatever scale that we can estimate a TPD function (species/population/community; see Carmona et al., 2016, 2019). In summary:

Functional richness (FRic) is the amount of functional volume occupied by a species, population or community. Therefore, a species with great variability in trait values between their individuals will have a higher FRic value than a species with low variability.

Functional evenness (FEve) reflects the evenness of the abundance of trait values in the analysed unit. Imagine two communities with the same functional richness (the same degree of variability in traits between their individuals). Suppose that some trait values are much more abundant than others in the first species, whereas all the trait values have a similar abundance in the second species: FEve for the second species should be higher than FEve for the first species.

Functional divergence (FDiv) reflects the distribution of abundances within the functional trait volume occupied by the analysed unit. A species in which the most abundant trait values are away from the center of gravity of the functional space –e.g. individuals are either very small, or very large– will display high FDiv, whereas a species in which the most abundant trait values are similar –e.g. most individuals having intermediate size– will have low FDiv .

The function `REND` calculates these three indices using the TPDs's and TPDc's of populations/species and communities, respectively. The function works with one or more dimensions, thus expanding the framework presented in Mason et al. (2005) to multiple traits without modifying its original rationale (see Chapter 5 in the reference book), based on probabilities. Let's apply the `REND` function to the `TPDc_result` object containing the TPDc of our 10 plots along the gradient:

```
REDPlots <- REND(TPDc = TPDc_result)
```

```
## Calculating FRichness of communities

## Calculating FEvenness of communities

## Calculating FDivergence of communities
```

`REDPlots` contains the FRic, FEve and FDiv values of all the communities (plots), considering the two traits simultaneously. For example, the values of functional richness of the plots are:

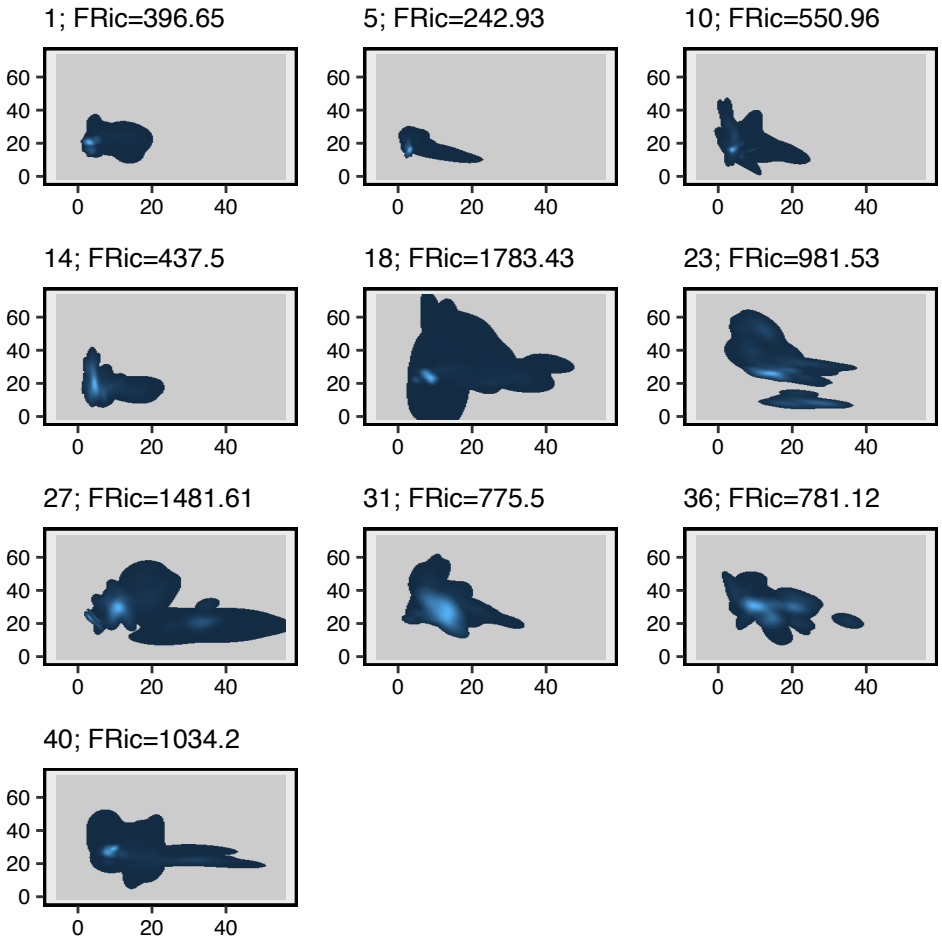
```
REDPlots$communities$FRichness
```

```
##         1         5        10        14        18        23        27        31
##  396.6500  242.9347  550.9625  437.4974 1783.4319  981.5325 1481.6148  775.5038
##        36        40
##  781.1174 1034.2042
```

For illustration purposes, let's plot the respective TPDc functions, along with their FRic values:

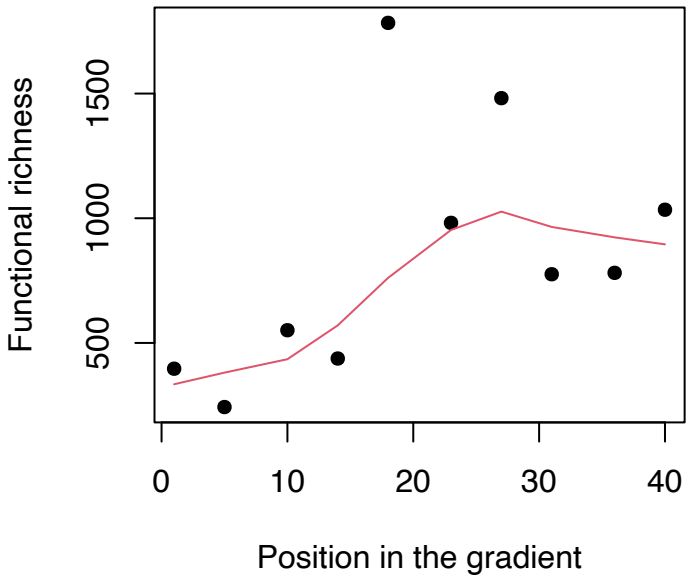
```
plotTPD(TPD = TPDc_result, leg.text = paste0(names(REDPlots$communities$FRichness),
  "; FRic=", round(REDPlots$communities$FRichness, 2)))
```

```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



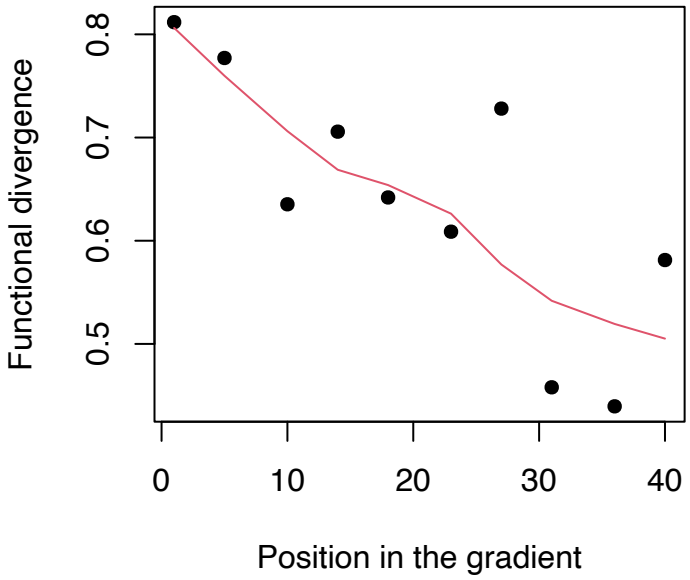
Plot "18" has the community that occupies the largest proportion of the functional space, whereas plot "5" is the one with the lowest value of functional richness. Some communities occupy a portion of the functional space that is distinctly different from other communities (e.g., communities 14 and 23), while other communities are more similar (e.g. communities 1 and 5). Now, we can examine if functional richness maximizes in the middle of the gradient, as it seemed before:

```
plot(REDPlots$communities$FRichness ~ unique(traits$Plot), pch = 16,
     xlab = "Position in the gradient", ylab = "Functional richness")
lines(lowess(unique(traits$Plot), REDPlots$communities$FRichness), col=2)
```



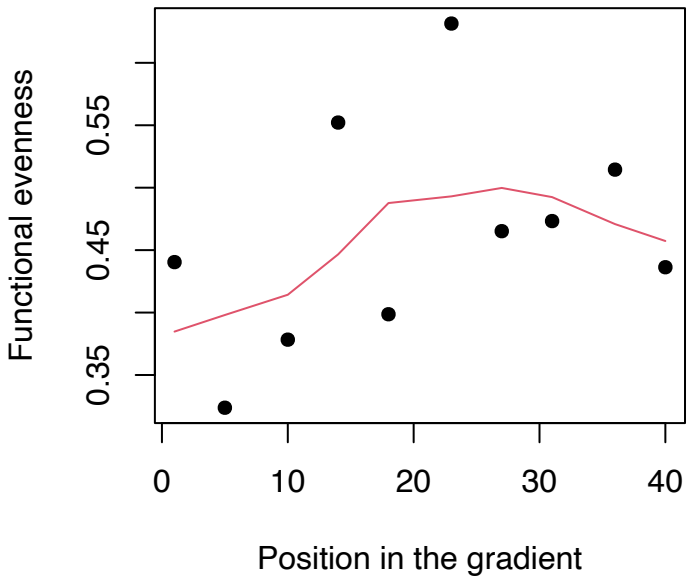
Functional richness increases at the beginning of the gradient and then reaches a plateau. What about the distribution of trait values within communities? It looks as if the most abundant trait values are closer to the boundaries of the distribution in the least productive end of the gradient, whereas the peak seems to me more "centered" in the lowest end of the gradient. This can be analysed using functional divergence:

```
plot(REDPlots$communities$FDivergence ~ unique(traits$Plot), pch = 16,
     xlab = "Position in the gradient", ylab = "Functional divergence")
lines(lowess(unique(traits$Plot), REDPlots$communities$FDivergence), col=2)
```



Functional evenness is a bit more tricky to predict from the plots, but one would say that it is lower in the less productive sites, where it seems as if the distribution is very "peaky", and increases as we go down. Let's see:

```
plot(REDPlots$communities$FEvenness ~ unique(traits$Plot), pch = 16,
     xlab = "Position in the gradient", ylab = "Functional evenness")
lines(lowess(unique(traits$Plot), REDPlots$communities$FEvenness), col=2)
```



6.6.7 Beta diversity (function `dissim`)

We have already explored the use of `dissim` to estimate the dissimilarity between species. Interestingly, since TPD functions are conceptually the same regardless of the ecological scale, the same function can be used to estimate the dissimilarity in trait space between communities (i.e. beta diversity, see Chapter 5 in the reference book). Our visual exploration of the plots suggested that communities at the low-productivity end of the gradient are very dissimilar to the communities at high-productivity end of the gradient. In general, it seems as if communities which are closer in the productivity gradient have a smaller functional dissimilarity, possibly because they experience similar environmental conditions. To explore this we first need to estimate the functional dissimilarity between communities:

```
dissim_comm <- dissim(TPDc_result)
```

```
## Calculating dissimilarities between 10 communities. It might take some time
```

```
round(dissim_comm$communities$dissimilarity, 3)
```

```
##         1     5    10    14    18    23    27    31    36    40
## 1   0.000 0.567 0.560 0.527 0.754 0.875 0.755 0.745 0.797 0.768
## 5   0.567 0.000 0.620 0.696 0.924 0.988 0.907 0.953 0.982 0.958
## 10  0.560 0.620 0.000 0.317 0.878 0.968 0.873 0.918 0.933 0.920
## 14  0.527 0.696 0.317 0.000 0.846 0.958 0.847 0.846 0.899 0.874
## 18  0.754 0.924 0.878 0.846 0.000 0.717 0.587 0.697 0.697 0.452
## 23  0.875 0.988 0.968 0.958 0.717 0.000 0.682 0.587 0.584 0.677
## 27  0.755 0.907 0.873 0.847 0.587 0.682 0.000 0.470 0.481 0.511
## 31  0.745 0.953 0.918 0.846 0.697 0.587 0.470 0.000 0.367 0.596
## 36  0.797 0.982 0.933 0.899 0.697 0.584 0.481 0.367 0.000 0.561
## 40  0.768 0.958 0.920 0.874 0.452 0.677 0.511 0.596 0.561 0.000
```

Then, we can make a matrix of spatial distance between our plots:

```
spatialDiss<- dist(unique(traits$Plot))
spatialDiss
```

```
##     1  2  3  4  5  6  7  8  9
## 2   4
## 3   9  5
## 4  13  9  4
## 5  17 13  8  4
## 6  22 18 13  9  5
## 7  26 22 17 13  9  4
## 8  30 26 21 17 13  8  4
## 9  35 31 26 22 18 13  9  5
## 10 39 35 30 26 22 17 13  9  4
```

Finally, we can plot the relationship between the two distances:

```
dev.off()
```

```
## null device
##           1
```

```
plot(spatialDiss, as.dist(dissim_comm$communities$dissimilarity), pch=16,
     xlab = "Spatial distance", ylab = "Functional beta diversity")
lines(lowess(spatialDiss, as.dist(dissim_comm$communities$dissimilarity)), col=2)
```

Community 5 and community 14 are rather dissimilar (0.696), despite they occupy a very similar part of the functional space. This is mostly due to differences in the shape of the two probabilistic distribution of traits, rather than to differences in the parts of the functional space occupied by each community. The other two elements of the `dissim_comm` object reflect this fact: dissimilarity between two TPD's can be decomposed into two complementary components. The first component is due to dissimilarities in the relative abundances of traits shared by the communities (as in the example between community 5 and community 14), whereas the other is due to the sections of the functional space that are exclusively occupied by one of the communities, but not by the other.

```
round(dissim_comm$communities$P_shared, 3)
```

```
##        1     5    10    14    18    23    27    31    36    40
## 1     NA 0.899 0.895 0.860 0.797 0.160 0.441 0.622 0.391 0.524
## 5  0.899    NA 1.000 0.971 0.747 0.022 0.076 0.145 0.049 0.128
## 10 0.895 1.000    NA 0.925 0.816 0.140 0.428 0.502 0.378 0.519
## 14 0.860 0.971 0.925    NA 0.898 0.088 0.432 0.368 0.280 0.551
## 18 0.797 0.747 0.816 0.898    NA 0.629 0.941 0.991 0.976 0.970
## 23 0.160 0.022 0.140 0.088 0.629    NA 0.642 0.586 0.709 0.525
## 27 0.441 0.076 0.428 0.432 0.941 0.642    NA 0.841 0.878 0.968
## 31 0.622 0.145 0.502 0.368 0.991 0.586 0.841    NA 0.901 0.973
## 36 0.391 0.049 0.378 0.280 0.976 0.709 0.878 0.901    NA 0.951
## 40 0.524 0.128 0.519 0.551 0.970 0.525 0.968 0.973 0.951    NA
```

```
round(dissim_comm$communities$P_non_shared, 3)
```

```
##        1     5    10    14    18    23    27    31    36    40
## 1     NA 0.101 0.105 0.140 0.203 0.840 0.559 0.378 0.609 0.476
## 5  0.101    NA 0.000 0.029 0.253 0.978 0.924 0.855 0.951 0.872
## 10 0.105 0.000    NA 0.075 0.184 0.860 0.572 0.498 0.622 0.481
## 14 0.140 0.029 0.075    NA 0.102 0.912 0.568 0.632 0.720 0.449
## 18 0.203 0.253 0.184 0.102    NA 0.371 0.059 0.009 0.024 0.030
## 23 0.840 0.978 0.860 0.912 0.371    NA 0.358 0.414 0.291 0.475
## 27 0.559 0.924 0.572 0.568 0.059 0.358    NA 0.159 0.122 0.032
## 31 0.378 0.855 0.498 0.632 0.009 0.414 0.159    NA 0.099 0.027
## 36 0.609 0.951 0.622 0.720 0.024 0.291 0.122 0.099    NA 0.049
## 40 0.476 0.872 0.481 0.449 0.030 0.475 0.032 0.027 0.049    NA
```

As we expected, `$P_shared` accounts for most of the dissimilarity between community 5 and community 14. The functional volume occupied by 5 is a subset of the space occupied by community 14, therefore `$P_shared` is close to 1. Conversely, community 5 and community 23 have a dissimilarity of 0.988, because they almost do not share any part of the trait space. As a consequence, the dissimilarity between community 5 and community 23 is mostly accounted by `$P_non_shared`.

6.6.8 Functional redundancy of communities (`redundancy`)

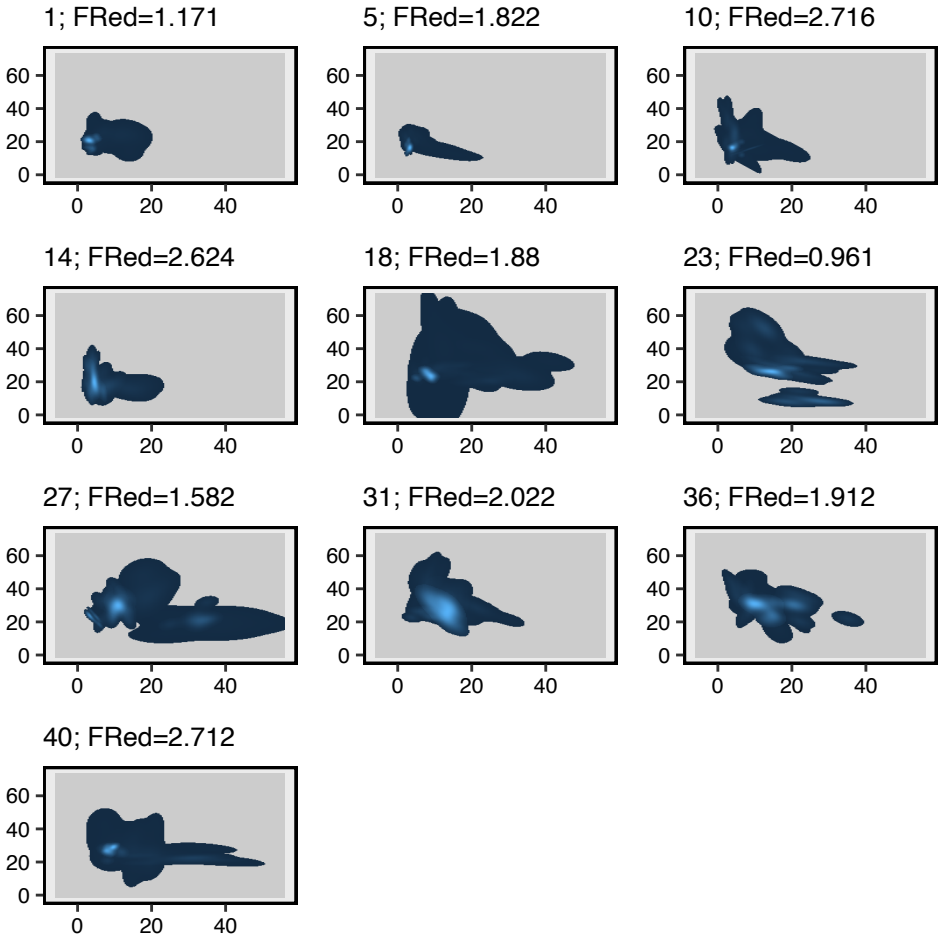
We consider that two species are redundant in a community when they occupy the same parts of the functional space, which means that they have the same trait values. Accordingly, if there are two species that are totally redundant, the loss of one of them will not reduce the functional volume occupied by the community (which is indicated by *functional richness*. In reality, communities are usually composed of more than two species, and therefore the concept of redundancy involves the simultaneous consideration of the functional volumes occupied by each species.

The method that we implement in the function `redundancy`, evaluates the number of species that occupy each cell of the grid defining the functional trait space occupied by the community. Afterwards, it performs a weighted mean for the whole community, using the probability associated to each cell (which are contained in TPDc) as the weighting factor. This gives the average number of species that are present in each cell; if we subtract 1 from this number, we get the average number of redundant species in the community as a whole, that is, the average number of species that could be lost in the community without reducing its functional volume. Let's explore the redundancy of our communities:

```
FRed_result <- redundancy(TPDc = TPDc_result)

plotTPD(TPD = TPDc_result, leg.text = paste(names(FRed_result$redundancy),
  round(FRed_result$redundancy, 3), sep="; FRed="))
```

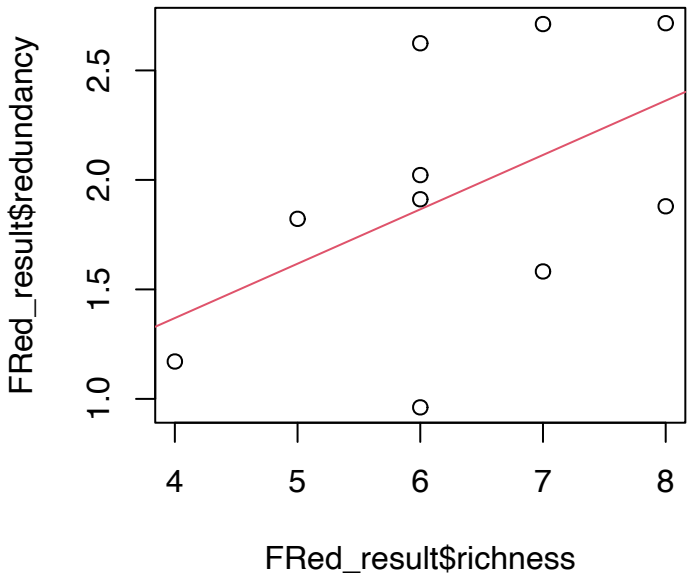
```
## Be patient, in the 2-dimensional case, plots can take some time.
## If you think it takes too long, consider reducing the number of plots using 'whichPlot'

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```

```
## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).

## Warning: Removed 796 rows containing missing values (geom_raster).
```



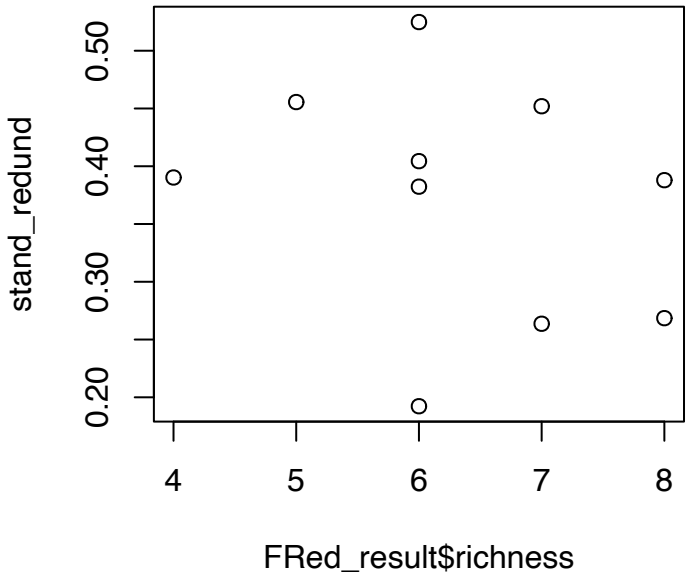
`redundancy` is unrelated to the functional volume occupied by each community. Actually, we would need to see which species are occupying each community to better visualize how it works. It is important to be aware that, due to the way it is conceived (the number of species you could remove on average without reducing the functional volume occupied by the community), redundancy is positively correlated with species richness:

```
plot(FRed_result$redundancy ~ FRed_result$richness)
abline(lm(FRed_result$redundancy ~ FRed_result$richness), col=2)
```



Indeed, the maximum value that redundancy can reach is Richness-1 (since redundancy when there is only 1 species must be 0), so that one can standardize redundancy values value by dividing them by richness - 1

```
stand_redund <- FRed_result$redundancy / (FRed_result$richness - 1)
plot(stand_redund ~ FRed_result$richness)
```



7 – Null models

Here, we will illustrate the use of *null models* to test ecological hypotheses about the *assembly of communities*. Note that we will focus exclusively on approaches that involve trait-based functional measures, i.e. we leave out the whole approach that relies only on species composition data (co-occurrence patterns), which has its own large literature regarding the indices and null models. As we mention repeatedly in Chapter 7 of the reference book, to make an adequate null model we have to first consider carefully what is the process of interest (are we interested in *environmental filtering*? are we interested in the effect of *biotic interactions*?). Based on what we want to test, we need to take a series of analytic decisions that should be considered with care, including the spatial scale, the indices that we will use and the reference species pool. Here, we will show different examples of null models aimed to examine different ecological questions. Though we do not provide a thorough examination of potential null models [readers can check the existing literature for that, including Hardy (2008), de Bello et al.](2012), Götzenberger et al. (2016), Mason et al. (2013), we try to show examples that would help readers understand how to perform a null model analysis in a variety of situations.

7.1 Data

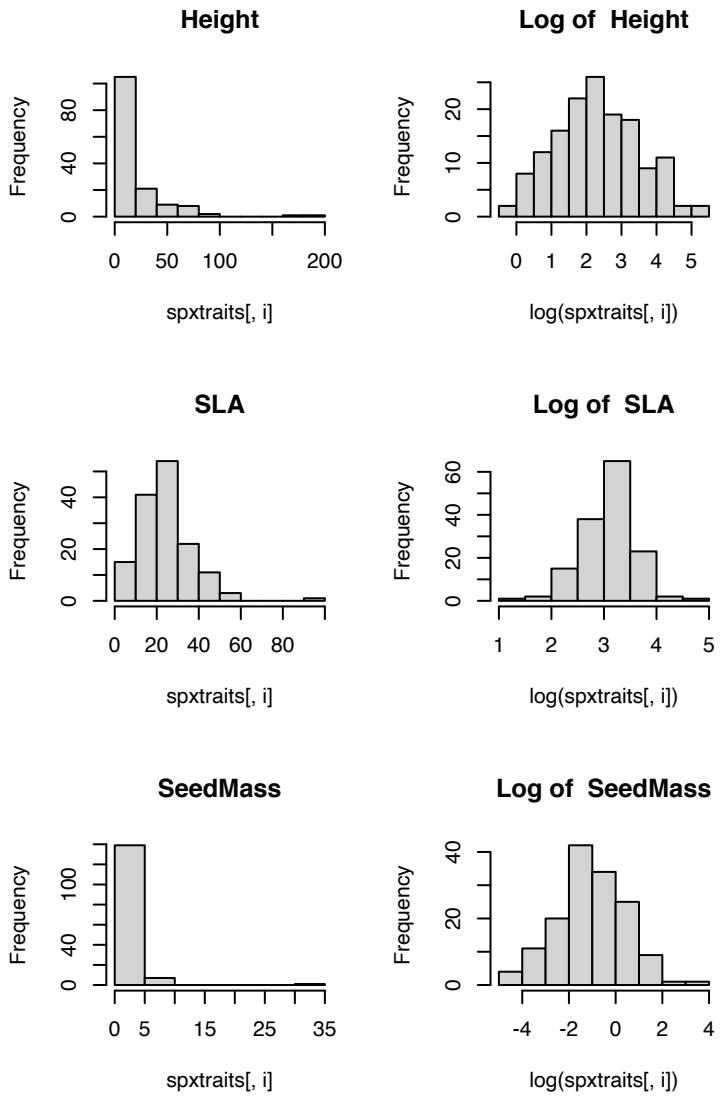
We will use data from Carmona et al. (2015a; 2012). In these papers, the authors sampled Mediterranean grassland communities with two different levels of productivity (determined by the position of the plots along slopes) and four different levels of grazing intensity (from grazing abandonment to almost constant presence of domestic grazers). The authors determined species composition in 8-9 plots per productivity and grazing level combination, for a total of 67 sites. They also measured three functional traits (vegetative height, specific leaf area and seed mass) in all species in the study area.

Let us start by loading the data:

```
spxtraits <- read.table(here::here("data", "chapter7", "SpxtTraits.txt"), row.names = 1, hea
plotxsp <- read.table(here::here("data", "chapter7","PlotxSp.txt"), row.names = 1, header =
plotxenv <- read.table(here::here("data", "chapter7","Environment.txt"), row.names = 1, head
```

First of all, let's take a look at the traits. It is very common that some of the traits we are using (in particular seed mass and plant height) have a very skewed distribution (i.e. some species have very large values while most species have relatively small values). We can make histograms of each trait both in the original scale and after log-transformation to see this:

```
par(mfrow = c(3, 2))
for (i in 1:ncol(spxtraits)) {
  hist(spxtraits[, i], main = colnames(spxtraits)[i])
  hist(log(spxtraits[, i]), main = paste("Log of ", colnames(spxtraits)[i]))
}
```



We can see how the distribution of all traits has become more "normal" after log-transforming. We will work with the log-transformed version of the traits from now on:

```
spxtraits <- log(spxtraits)
```

The next object we have is `plotxsp`, which includes the relative abundance of each species (147 species, in columns) in each of the plots that were sampled (67 plots, in rows):

```
plotxsp[24:30, 1:5]
```

```
##        Agrostis_castellana Agrostis_pourretii Aira_cupaniana
## CS007          0.00000000                   0     0.00000000
## CS008          0.00000000                   0     0.00000000
## PEH001         0.07738095                   0     0.00000000
## PEH002         0.26300985                   0     0.00000000
## PEH003         0.10034602                   0     0.00000000
## PEH004         0.05285714                   0     0.00000000
## PEH005         0.02951192                   0     0.01702611
##        Alopecurus_arundinaceus Alopecurus_geniculatus
## CS007                        0                      0
## CS008                        0                      0
```

```
## PEH001                    0                    0
## PEH002                    0                    0
## PEH003                    0                    0
## PEH004                    0                    0
## PEH005                    0                    0
```

```
dim(plotxsp)
```

```
## [1]  67 147
```

Finally, the object `plotxenv` contains the environmental information, in this case productivity ("Low" or "High") and grazing level (from 1 being very low grazing to 4 being high grazing intensity) of each plot:

```
head(plotxenv)
```

```
##       Grazing Productivity
## NS001       4          Low
## NS002       4          Low
## NS003       4          Low
## NS004       4          Low
## NS005       4          Low
## NS006       4          Low
```

```
dim(plotxenv)
```

```
## [1] 67  2
```

7.2 Required packages and additional functions

Here are the packages that we need to load to perform all the analyses that we show below. Remember that they should be installed in your computer or you will not be able to load them:

```
library(FD)
library(picante)
library(DarkDiv)
```

```
## Warning: package 'DarkDiv' was built under R version 4.0.2
```

In addition, to estimate the mean pairwise dissimilarity, we will use the function `melodic` (from de Bello et al., 2016):

```
source(here::here("data", "chapter7","melodic.R"))
```

7.3 Introduction to randomizations

The basic idea behind null models is relatively simple: we want to see if some observed pattern of diversity (in our case, of functional diversity) differs from what we could expect by chance. To do this, the first thing we need to do is to be able to estimate the index of functional diversity we want to examine. We already know how to do this thanks to Chapter 5 of the reference book and to R material Ch 5. The next step is to create "random" samples following some rule that redistributes the observed data. The term "null model" implies that this redistribution of data reflects the null hypothesis that the observed pattern of diversity is due to chance (Gotelli and Graves, 1996; Götzenberger et al., 2012). To do this, the most common alternative is to randomize our data (for example, our plots x species matrix) and calculate the same index of diversity for those randomized data. If we repeat this process many times, we will have a value of this index for each repetition, i.e. we will have a set of expected values under random assembly, which allows us to build a "null distribution" with which we can compare the observed value. The comparison of observed patterns vs. expected ones is often done by means of the "Standardized Effect Size", *SES*, which is calculated by subtracting the mean of the null distribution from the observed value and dividing the result by the standard deviation of the null distribution (see Chapter 7 in the reference book).

Then, the main difficulty here is how to perform the necessary randomization, that allows us to randomize the pattern of interest while keeping all other aspects of the data unchanged. Let's illustrate what we mean with a very simple example. The `picante` package includes a dataset called `phylocom`, which contains a sites x species matrix (`phylocom$sample`) and a species x traits matrix (`phylocom$traits`).

```
data(phylocom)
phylocom$sample
```

```
##         sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19 sp2 sp20 sp21 sp22
## clump1    1    0    0    0    0    0    0    0    0    0   1    0    0    0
## clump2a   1    2    2    2    0    0    0    0    0    0   1    0    0    0
## clump2b   1    0    0    0    0    0    0    2    2    2   1    2    0    0
## clump4    1    1    0    0    0    0    0    2    2    0   1    0    0    0
## even      1    0    0    0    1    0    0    1    0    0   0    0    1    0
## random    0    0    0    1    0    4    2    3    0    0   1    0    0    1
##         sp24 sp25 sp26 sp29 sp3 sp4 sp5 sp6 sp7 sp8 sp9
## clump1     0    0    0    0   1   1   1   1   1   1   0
## clump2a    0    0    0    0   1   1   0   0   0   0   2
## clump2b    0    0    0    0   1   1   0   0   0   0   0
## clump4     0    2    2    0   0   0   0   0   0   0   1
## even       0    1    0    1   0   0   1   0   0   0   1
## random     2    0    0    0   0   0   2   0   0   0   0
```

`phylocom$sample` contains information about species abundance. We can make things simpler by now, by working first only with a presence absence matrix. This can be done as:

```
phylocomPA <- replace(phylocom$sample, phylocom$sample > 0, 1)
```

In addition, there are more species in the trait matrix than in the samples one. To make our life easier, lets keep only the traits of the species present in the samples:

```
phylocomTraits <- phylocom$traits[colnames(phylocomPA), ]
head(phylocomTraits)
```

```
##      traitA traitB traitC traitD
## sp1       1      1      1      0
```

```
## sp10      1      3      2      1
## sp11      2      3      3      1
## sp12      2      3      4      1
## sp13      2      4      1      1
## sp14      2      4      2      1
```

Below we will first show how randomizations are done, and then we will show some existing algorithms that can speed up the process. Finally, we will perform some ad-hoc randomizations not yet available in R.

7.4 Unconstrained randomizations by hand

Most classical null models are based on randomizing some part of the sample matrix while leaving some other things unchanged. We know that the Faith's PD index is related to species richness (see Chapter 5 in the reference book). Although this makes sense, sometimes we want to know if a particular site has a higher or lower Faith's PD than what we could expect given its species richness. In other cases, we might want to compare the levels of Faith's PD of sites with large differences in species richness. We could make a null model, in which we keep fixed the number of species in our sites, randomize the identity of species and then estimate Faith's PD. If we do this many times and estimate a SES value, then we can see if Faith's PD in each site is higher (positive SES) or lower (negative SES) than expected given the number of species. Let's create such a randomization step by step.

First, we need to count how many species are present in each site (in each row). This can be done like this:

```
SpRichness <- rowSums(phylocomPA)
SpRichness
```

```
##  clump1 clump2a clump2b  clump4    even  random
##       8       8       8       8       8       8
```

Cool. All our sites have the same number of species, so they only differ in which species are present. The second step here would be to randomly assign, to each site, 8 species from the species pool (from all the species in the matrix). We can first create an empty matrix that we will initially fill with zeros, like this:

```
randomizedPA <- phylocomPA
randomizedPA[, ] <- 0
```

Then, we successively fill the still empty cells with the results of the randomization. For each site (let's start by the first site, "clump1"), we can randomly select 8 species:

```
randomSP <- sample(colnames(randomizedPA), size = SpRichness[1])
randomSP
```

```
## [1] "sp2"  "sp18" "sp1"  "sp12" "sp4"  "sp14" "sp24" "sp17"
```

So, in the site "clump1", we will include the 8 species contained in `randomSP`:

```
randomizedPA["clump1", randomSP] <- 1
randomizedPA
```

```
##         sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19 sp2 sp20 sp21 sp22
## clump1    1    0    0    1    0    1    0    1    1    0   1    0    0    0
## clump2a   0    0    0    0    0    0    0    0    0    0   0    0    0    0
## clump2b   0    0    0    0    0    0    0    0    0    0   0    0    0    0
## clump4    0    0    0    0    0    0    0    0    0    0   0    0    0    0
## even      0    0    0    0    0    0    0    0    0    0   0    0    0    0
## random    0    0    0    0    0    0    0    0    0    0   0    0    0    0
##         sp24 sp25 sp26 sp29 sp3 sp4 sp5 sp6 sp7 sp8 sp9
## clump1     1    0    0    0   0   1   0   0   0   0   0
## clump2a    0    0    0    0   0   0   0   0   0   0   0
## clump2b    0    0    0    0   0   0   0   0   0   0   0
## clump4     0    0    0    0   0   0   0   0   0   0   0
## even       0    0    0    0   0   0   0   0   0   0   0
## random     0    0    0    0   0   0   0   0   0   0   0
```

Instead of rewriting similar lines of code eight times, we can make a loop to repeat the process across sites. Note that we make one "trick", which is using the function `set.seed()` to make the results repeatable among users (so that if you run the code in your own R session, you should get the same results as here; cool!):

```
set.seed(1)
randomizedPA <- phylocomPA
randomizedPA[,] <- 0 #Reset the matrix to be filled
for (i in 1:nrow(phylocomPA)) {#For each row in the matrix (for each site)
  #select randomly, as many species as the species richness of the site:
  randomSP <- sample(colnames(randomizedPA), size = SpRichness[i])
  randomizedPA[i, randomSP] <-1 #Fill the cells of those species with a 1
}
randomizedPA
```

```
##         sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19 sp2 sp20 sp21 sp22
## clump1    1    1    0    1    0    0    1    0    0    0   1    0    0    1
## clump2a   1    0    0    0    0    0    1    0    0    1   0    0    0    1
## clump2b   0    0    0    0    1    0    0    0    1    0   0    0    0    1
## clump4    1    1    1    0    0    1    0    0    0    1   0    1    0    0
## even      0    0    0    0    0    1    0    0    0    1   0    1    0    0
## random    0    0    0    0    0    1    1    1    0    1   0    1    0    0
##         sp24 sp25 sp26 sp29 sp3 sp4 sp5 sp6 sp7 sp8 sp9
## clump1     0    0    0    1   0   0   0   0   0   0   1
## clump2a    0    0    0    0   1   0   1   1   1   0   0
## clump2b    1    0    0    0   1   0   1   1   0   0   1
## clump4     1    0    0    0   0   1   0   0   0   0   0
## even       1    0    0    0   0   1   1   0   1   0   1
## random     0    0    0    0   1   0   0   1   0   0   1
```

The object `randomizedPA` now contains 8 species in each site, but the identities of the species are different than the original matrix. We can estimate Faith's PD (see R material 5 in the original matrix and the randomized one and compare them:

```
trait.d <- gowdis(phylocomTraits)
tree.traits <- hclust(trait.d, "average")
ultra.tree.traits <- as.phylo(tree.traits)
faithFD <- pd(phylocomPA, ultra.tree.traits)

obsFaith <- pd(phylocomPA, ultra.tree.traits)$PD
simFaith <- pd(randomizedPA, ultra.tree.traits)$PD

data.frame(obsFaith, simFaith)
```

```
##    obsFaith simFaith
## 1 0.9080882 1.171977
## 2 1.1745024 1.267095
## 3 1.1745024 1.306447
## 4 0.7179330 1.364317
## 5 1.2161690 1.223113
## 6 1.4268172 1.308762
```

This way, we have ran the first iteration of our null model. We can see that observed Faith's PD values are smaller than simulated ones in some cases (e.g. for the first site), and higher in other cases (e.g. for the last site). However, we cannot be sure if this is due to just randomness, since we only made an iteration. This kind of null models require many iterations in order to be able to get a reliable null distribution of Faith's PD values for each site. Here we show how this can be done, for example, 100 times (note that generally more iterations are better, but let's keep it small for these examples):

```
set.seed(1)
numberReps <- 100
#Lets create a matrix to store results from each iteration (one column per iteration)
resultsRandom <- matrix(NA, nrow = nrow(phylocomPA), ncol = numberReps,
                        dimnames = list(rownames(phylocomPA),
                                        paste0("Sim.", 1:numberReps)))
for(rep in 1:numberReps){
  randomizedPA <- phylocomPA
  randomizedPA[,] <- 0
  for(i in 1:nrow(phylocomPA)){
    randomSP <- sample(colnames(randomizedPA), size = SpRichness[i])
    randomizedPA[i, randomSP] <-1 #Fill the cells of those species with a 1
  }
  simFaith <- pd(randomizedPA, ultra.tree.traits)$PD
  resultsRandom[, rep] <- simFaith
}
obsFaith <- pd(phylocomPA, ultra.tree.traits)$PD
```

That should have been pretty fast, right? Now we have `resultsRandom` containing in each column the Faith's PD value of each iteration. Now we really can see if the observed values are smaller or larger than expected for the same number of species. For this, lets calculate, for each site, the average of the simulated Faith's PD values:

```
meanNull <- rowMeans(resultsRandom)
data.frame(obsFaith, meanNull)
```

```
##          obsFaith meanNull
## clump1  0.9080882 1.248222
## clump2a 1.1745024 1.278257
## clump2b 1.1745024 1.281403
## clump4  0.7179330 1.260979
## even    1.2161690 1.276960
## random  1.4268172 1.270937
```

The first thing to notice is that all values in `meanExpected` are very similar. This should not be surprising, since they reflect the mean expected value of Faith's PD for communities with eight species, and all our sites have eight species! Hence the mean randomized values, particularly when keeping richness fixed, is generally quite constant (de Bello et al., 2011). Notice also that we keep richness fixed, but that all of the 29 species in the dataset will have the same probability of being "selected" in the randomizations as one of the eight members

in a given site. Repeating this sampling procedure will therefore "average out" the variation from single iterations across the sites.

We can calculate the *Effect Size* which is the difference between the observed value and the expected one:

```
ES <- obsFaith - meanNull
ES
```

```
##      clump1      clump2a      clump2b      clump4        even      random
## -0.34013369 -0.10375452 -0.10690056 -0.54304565 -0.06079075  0.15588012
```

We can see now that some sites have lower than expected Faith's PD (e.g. "clump4"), whereas the "random" site has higher than expected Faith's PD. The effect size is expressed in the same scale as Faith's PD. The last step is to divide the effect size by the standard deviation of the null distribution of Faith's PD values of each site, i.e. the Effect Size ("ES") becomes the Standardized Effect Size ("SES"). This step is not particularly important here, since all sites have the same species richness, but it is better when we want to compare sites with different number of species:

```
sdNull <- apply(resultsRandom, 1, sd)
SES <- ES / sdNull
SES
```

```
##     clump1    clump2a    clump2b     clump4       even     random
## -3.2395515 -1.1076764 -0.9800692 -4.6833590 -0.6238642  1.4987019
```

A good thing of SES values is that they are always expressed in the same scale: standard deviations. This makes interpretation rather straightforward once we get used to it. For example, SES in site "clump1" is -3.24. This means that the observed value is 3.24 standard deviations smaller than the average of the null values. As a general rule of thumb, for a two-tailed test, SES values smaller than -2 (or larger than 2) indicate that the observed value is significantly smaller (or larger) than expected by chance, so that we can use them (with some caution) not only to assess differences between our sites, but also whether there is trait underdispersion (SES $<$ -2) or overdispersion (SES $>$ 2) within sites. For a one-tailed test the threshold value is 1.56. Note however that this logic only applies in the SES values follow a normal distribution, which is not always the case! In any case, SES values are also interesting when they are not further than 2 sd units from zero! For example, imagine you have sampled a series of sites along an aridity gradient because you are are interested in examining whether variability in traits is restricted when aridity increases. In that case, you might want to regress SES values against aridity values, expecting to observe a negative relationship. Perhaps the sites under the most arid conditions have SES values smaller than -2, but simply assessing if the slope of the SES~aridity relationship is indeed negative is a very interesting question on its own (but see de Bello et al. (2011) for some cases when a null-model is actually not needed, particularly for indices not so influenced by species richness).

7.5 Randomization functions

Luckily, you do not need to make your null models by hand all the time. There are some functions in R that allow us to make the necessary randomizations. One of them is the `randomizeMatrix` function from the `picante` package; another one is `commsim` in `vegan`. We will illustrate the use of `randomizeMatrix` for simplicity, as it allows for making several different interesting null models; we will explore some of them later. By now, let's take a look at the null model called "richness". This null model, according to the `randomizeMatrix` help "Randomize community data matrix abundances within samples (maintains sample species

richness)", which sounds pretty much like what we have just done above. Let's check it out it:

```
richnessPA <- randomizeMatrix(samp = phylocomPA, null.model = "richness",
                              iterations = 1)
richnessPA
```

```
##         sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19 sp2 sp20 sp21 sp22
## clump1    0    1    0    1    1    0    0    1    0    0   1    0    0    1
## clump2a   1    1    0    0    0    1    0    0    0    1   0    0    0    0
## clump2b   1    0    0    0    0    1    1    0    0    1   1    0    1    0
## clump4    0    0    0    0    1    0    1    0    1    0   0    1    0    0
## even      0    0    0    0    1    1    0    1    0    1   1    0    0    1
## random    1    0    1    0    0    0    1    0    1    0   0    1    0    1
##         sp24 sp25 sp26 sp29 sp3 sp4 sp5 sp6 sp7 sp8 sp9
## clump1     0    0    1    0   0   0   1   0   0   0   0
## clump2a    0    0    1    1   0   1   0   1   0   0   0
## clump2b    0    0    1    0   0   0   0   0   1   0   0
## clump4     0    0    0    0   1   0   0   1   1   0   1
## even       0    0    0    0   0   0   1   0   0   0   1
## random     0    1    0    0   0   0   0   1   0   0   0
```

```
rowSums(richnessPA)
```

```
##  clump1 clump2a clump2b  clump4    even  random
##       8       8       8       8       8       8
```

As we can see, the `randomizeMatrix` function has delivered what we wanted: a matrix where the number of species in each site is fixed, but the identity of the species present in each site changes. We could now repeat our SES estimations with fewer lines of code (and smaller chance of errors):

```
numberReps <- 100
#Lets create a matrix to store results from each iteration (one column per iteration)
resultsRandom <- matrix(NA, nrow = nrow(phylocomPA), ncol = numberReps,
                        dimnames = list(rownames(phylocomPA),
                                        paste0("Sim.", 1:numberReps)))
for(rep in 1:numberReps){
  randomizedPA <- randomizeMatrix(samp = phylocomPA, null.model = "richness",
                                  iterations = 1)
  simFaith <- pd(randomizedPA, ultra.tree.traits)$PD
  resultsRandom[, rep] <- simFaith
}
obsFaith2 <- pd(phylocomPA, ultra.tree.traits)$PD
meanNull2 <- rowMeans(resultsRandom)
ES2 <- obsFaith2 - meanNull2
sdNull2 <- apply(resultsRandom, 1, sd)
SES2 <- ES2 / sdNull2
data.frame(SES, SES2)
```

```
##                SES       SES2
## clump1  -3.2395515 -3.2523497
## clump2a -1.1076764 -0.8457599
## clump2b -0.9800692 -1.2479076
## clump4  -4.6833590 -5.2680034
## even    -0.6238642 -0.3893936
## random   1.4987019  1.5789303
```

Note that the results from our first attempt (left column) are not completely identical than those from our first attempt (right column). This is partly because we have used a very small number of repetitions (100), and partly because of the randomness associated to making null models. However, you can see how the two procedures have led to very similar results.

So, we have made our first null model and estimated our first set of SES values. However, this simple kind of null model, in which we fix the number of species in each community and select species completely randomly from the pool of species is problematic. This is because there are some aspects of community structure that we are not fixing. The most evident one is the frequency of each species in the dataset. Let's take the matrix from the last randomization we performed (`randomizedPA`) and examine how frequent each species is, and then compare that with the original dataset (`phylocomPA`):

```
colSums(randomizedPA)
```

```
##  sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19  sp2 sp20 sp21 sp22 sp24 sp25
##    2    3    2    4    1    1    0    4    1    2    2    1    1    3    1    2
## sp26 sp29  sp3  sp4  sp5  sp6  sp7  sp8  sp9
##    2    1    4    1    1    1    2    4    2
```

```
colSums(phylocomPA)
```

```
##  sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19  sp2 sp20 sp21 sp22 sp24 sp25
##    5    2    1    2    1    1    1    4    2    1    5    1    1    1    1    2
## sp26 sp29  sp3  sp4  sp5  sp6  sp7  sp8  sp9
##    1    1    3    3    3    1    1    1    3
```

As you can see, some species that were originally rare are much more frequent in the randomized dataset, whereas other species that were much more common have become much less frequent. This is problematic, since in most cases, the prevalence of species is something we do not want to modify (so that it should also be fixed in our null model). Note that in the kind of null model we have just explored we only fixed a single pattern (i.e. species richness), while all other things are allowed to vary among repetitions (e.g. the frequency of species).

7.6 The independent swap randomization algorithm

An alternative is to perform randomizations in which we really try to fix all aspects of community structure other than the one we are interested in changing. For example, the *independent swap algorithm* keeps constant both species richness and the total number of occurrences of each species. The algorithm basically works by "swapping" pairs of species between pairs of sites, with the restriction that each species is present in one site but not in the other. For example, in our matrix `phylocomPA`, sp1 is present in the site "even" and absent from the site "random", whereas sp12 is present in "random" but not in "even". The algorithm could swap these species, so that in the null model, sp1 would be present now if "random" and sp12 would be present in "even", but both the total number of species in "random" and "even" and the total number of occurrences of sp1 and sp12 would be unchanged (see Gotelli, 2000, for more details). The `randomizeMatrix` function allows us to implement this kind of null model:

```
indepswapPA <- randomizeMatrix(samp = phylocomPA, null.model = "independentswap")
indepswapPA
```

```
##         sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19 sp2 sp20 sp21 sp22
## clump1    1    1    1    0    0    0    0    0    0    0   1    0    0    1
## clump2a   1    1    0    1    0    0    0    1    1    0   1    0    0    0
## clump2b   1    0    0    0    0    0    1    0    1    0   1    0    1    0
## clump4    1    0    0    1    0    0    0    1    0    0   1    0    0    0
## even      1    0    0    0    0    1    0    1    0    1   0    1    0    0
## random    0    0    0    0    1    0    0    1    0    0   1    0    0    0
##         sp24 sp25 sp26 sp29 sp3 sp4 sp5 sp6 sp7 sp8 sp9
## clump1     0    1    0    0   1   0   0   0   0   0   1
## clump2a    0    0    0    0   1   1   0   0   0   0   0
## clump2b    1    0    0    0   0   1   1   0   0   0   0
## clump4     0    0    1    0   1   0   0   0   0   1   1
## even       0    0    0    1   0   0   1   0   0   0   1
## random     0    1    0    0   0   1   1   1   1   0   0
```

```
rowSums(indepswapPA)
```

```
##  clump1 clump2a clump2b  clump4    even  random
##       8       8       8       8       8       8
```

```
colSums(indepswapPA)
```

```
##  sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19  sp2 sp20 sp21 sp22 sp24 sp25
##    5    2    1    2    1    1    1    4    2    1    5    1    1    1    1    2
## sp26 sp29  sp3  sp4  sp5  sp6  sp7  sp8  sp9
##    1    1    3    3    3    1    1    1    3
```

```
colSums(phylocomPA)
```

```
##  sp1 sp10 sp11 sp12 sp13 sp14 sp15 sp17 sp18 sp19  sp2 sp20 sp21 sp22 sp24 sp25
##    5    2    1    2    1    1    1    4    2    1    5    1    1    1    1    2
## sp26 sp29  sp3  sp4  sp5  sp6  sp7  sp8  sp9
##    1    1    3    3    3    1    1    1    3
```

As you can see, now both the number of species in each site and the number of occurrences of each species across sites are fixed (so that sp1 occurs in five sites in both cases). However, the sites in which sp1 occurs differ between the original matrix and the `indepswapPA` matrix we just created. Let's estimate SES values for Faith's PD again using this new type of null model:

```
numberReps <- 100
#Lets create a matrix to store results from each iteration (one column per iteration)
resultsRandom <- matrix(NA, nrow = nrow(phylocomPA), ncol = numberReps,
                        dimnames = list(rownames(phylocomPA),
                                        paste0("Sim.", 1:numberReps)))
for(rep in 1:numberReps){
  randomizedPA <- randomizeMatrix(samp = phylocomPA, null.model = "independentswap")
  simFaith <- pd(randomizedPA, ultra.tree.traits)$PD
  resultsRandom[, rep] <- simFaith
}
obsFaith3 <- pd(phylocomPA, ultra.tree.traits)$PD
meanNull3 <- rowMeans(resultsRandom)
ES3 <- obsFaith3 - meanNull3
sdNull3 <- apply(resultsRandom, 1, sd)
SES3 <- ES3 / sdNull3
data.frame(SES, SES2, SES3)
```

```
##                SES       SES2       SES3
## clump1  -3.2395515 -3.2523497 -4.0249971
## clump2a -1.1076764 -0.8457599 -0.9396246
## clump2b -0.9800692 -1.2479076 -1.2227703
## clump4  -4.6833590 -5.2680034 -6.8761868
## even    -0.6238642 -0.3893936 -1.0262648
## random   1.4987019  1.5789303  1.6582286
```

As you can see, the results do not differ much from the ones we obtained before. However, now we are sure that the SES values that we have are not due to spurious changes in the total frequency of species. Notice however that using null models with a lot of constrains comes with some problems. For example, the 100 repetitions done above could be actually be too similar between them in a small dataset, simply because the number of potential combinations is reduced when introducing randomizations with more constrains. This is something that should be considered specifically in each particular study.

7.7 SES functions

You might have realized already that making null models often requires creating relatively complex scripts. While this can be a bit frustrating for beginners, one gets used to it with practice, and this offers you a lot of control over what you are doing. However, there are some functions that allow you to estimate SES values with a single call. Among these, there is a family of functions in the package `picante` that is very handy for these kind of calculations. The name of these functions starts with the letters *ses.*, which is the followed by the index of functional diversity that is randomized (e.g. `ses.mpd`, `ses.pd`, `ses.mntd`). They include an internal call to the `randomizeMatrix` function to randomize community composition following different criteria. Since we have been exploring Faith's PD so far, lets apply `ses.pd` to our small dataset:

```
numberReps <- 100
machine <- ses.pd(samp = phylocomPA, tree = ultra.tree.traits,
                  null.model="independentswap", runs = numberReps)
machine
```

```
##         ntaxa    pd.obs pd.rand.mean pd.rand.sd pd.obs.rank   pd.obs.z
## clump1      8 0.9080882     1.289671 0.09197686           1 -4.1486792
## clump2a     8 1.1745024     1.284648 0.08462126           9 -1.3016256
## clump2b     8 1.1745024     1.300858 0.08500909           9 -1.4863730
## clump4      8 0.7179330     1.295172 0.08511045           1 -6.7822291
## even        8 1.2161690     1.286375 0.10463173          21 -0.6709841
## random      8 1.4268172     1.282207 0.09719024          99  1.4879137
##           pd.obs.p runs
## clump1  0.00990099  100
## clump2a 0.08910891  100
## clump2b 0.08910891  100
## clump4  0.00990099  100
## even    0.20792079  100
## random  0.98019802  100
```

The resulting object (`machine`), contains a few columns, including the PD values observed in each sample (you can see that they are identical as the ones contained in `obsFaith3`). The column `pd.obs.z` contains the SES values (because the type of standardization applied for SES is also known as z-transformation). We can now compare them with those from our previous attempts:

```
SES4 <- machine$pd.obs.z
data.frame(SES, SES2, SES3, SES4)
```

```
##                SES       SES2       SES3       SES4
## clump1  -3.2395515 -3.2523497 -4.0249971 -4.1486792
## clump2a -1.1076764 -0.8457599 -0.9396246 -1.3016256
## clump2b -0.9800692 -1.2479076 -1.2227703 -1.4863730
## clump4  -4.6833590 -5.2680034 -6.8761868 -6.7822291
## even    -0.6238642 -0.3893936 -1.0262648 -0.6709841
## random   1.4987019  1.5789303  1.6582286  1.4879137
```

Can you see that SES4 results are very similar to the ones in SES3? This is because we are using the same method, and they differ only because we didn't make many repetitions and there is some inherent randomness associated to the process (i.e. a set of 100 iterations will never get numerically equal results to another set of 100 iterations). Actually, if you make a call to `set.seed` before estimating both SES3 and SES4 you should get exactly the same results, but we leave testing that up to you.

7.8 Randomizing the trait matrix

Despite its nice properties, the independent swap algorithm works only with presence/absence matrices. Sometimes we are also interested on the effect of the abundances of each species. When abundances enter into play, making null models becomes a bit more complicated. This is because there are even more properties that we can choose to keep fixed or not, such as the total summed abundance over a site, and the total abundance each species reaches when summed over the plots. So even more different ways to set up our null models! An alternative then is simply to shuffle the rows of our trait matrix, so that we randomly assign to each species the traits of another species, while we leave the community data completely untouched, thus preserving all its properties. What we effectively test now with our null model is that the species traits have nothing to do with the compositional differences across the sites (and underlying environmental conditions). Notice that by keeping the traits of the same species together, but changing the identity of the species they belong to, we are conserving the correlation between traits.

We will exemplify this using the `mpd` index, which, as we showed in R material Ch 5, can accommodate species abundances (as implemented in the `melodic` function). And we will use the `phylocom$sample` which contains species abundances as our samples x species matrix. As you might remember, mpd is based on species dissimilarities, so that we have to estimate them, using, for example, gower distances:

```
dissimObs<-gowdis(phylocom$traits)
```

Once we have this, we can estimate mpd:

```
mpdObs <- melodic(samp = phylocom$sample, dis = dissimObs,
                  type = "abundance")$abundance$mpd
```

```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
names(mpdObs) <- rownames(phylocom$sample)
mpdObs
```

```
##    clump1   clump2a   clump2b    clump4      even    random
## 0.2738095 0.4811828 0.3951613 0.2258065 0.3690476 0.4483025
```

Good. So, we can see that species in "clump2a" are more dissimilar between them than in "clump4" for example. Let's make a null model to estimate SES values. In this case, in each randomization, we have to recalculate the dissimilarity matrix by randomizing rows in the trait matrix. Note that it is important to randomize full rows, so that the combinations of the four traits that are observed for a given species stay together after the randomization and only the identity of the species that "receives" that set of traits changes. Otherwise, we would be altering the patterns of covariation between traits, which is not something we generally want to do. We can do the randomization, for example, like this:

```
traitsRand <- phylocom$traits[sample(1:nrow(phylocom$traits)),]
rownames(traitsRand) <- rownames(phylocom$traits)
```

So, we have shuffled the rows, and then reordered the names to get the same order as originally (this avoids problems with some functions that do not check whether species names are identical and in the same order in both trait and community data when estimating functional diversity indices). Finally, we can re-estimate the gower dissimilarity and mpd:

```
dissimRand <- gowdis(traitsRand)
mpdRand <- melodic(samp = phylocom$sample, dis = dissimRand,
                   type = "abundance")$abundance$mpd
```

```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
names(mpdRand) <- rownames(phylocom$sample)
data.frame(mpdObs, mpdRand)
```

```
##            mpdObs   mpdRand
## clump1  0.2738095 0.4523810
## clump2a 0.4811828 0.3776882
## clump2b 0.3951613 0.3561828
## clump4  0.2258065 0.3924731
## even    0.3690476 0.3988095
## random  0.4483025 0.4498457
```

So, this would correspond to a single iteration of our null model. Let's run it 100 times and estimate SES values!

```
numberReps <- 100
#Lets create a matrix to store results from each iteration (one column per iteration)
resultsRandom <- matrix(NA, nrow = nrow(phylocom$sample), ncol = numberReps,
                        dimnames = list(rownames(phylocom$sample),
                                        paste0("Sim.", 1:numberReps)))
for(rep in 1:numberReps){
  traitsRand <- phylocom$traits[sample(1:nrow(phylocom$traits)),]
  rownames(traitsRand) <- rownames(phylocom$traits)
  dissimRand <- gowdis(traitsRand)
  mpdRand<-melodic(samp = phylocom$sample, dis = dissimRand,
                   type = "abundance")$abundance$mpd
  resultsRandom[, rep] <- mpdRand
}
mpdObs <- melodic(samp = phylocom$sample, dis = dissimObs,
                  type = "abundance")$abundance$mpd
meanNull5 <- rowMeans(resultsRandom)
ES5 <- mpdObs - meanNull5
sdNull5 <- apply(resultsRandom, 1, sd)
SES5 <- ES5 / sdNull5
data.frame(SES, SES2, SES3, SES4, SES5)
```

```
##                SES       SES2       SES3       SES4       SES5
## clump1  -3.2395515 -3.2523497 -4.0249971 -4.1486792 -1.9507607
## clump2a -1.1076764 -0.8457599 -0.9396246 -1.3016256  0.9936421
## clump2b -0.9800692 -1.2479076 -1.2227703 -1.4863730 -0.2246218
## clump4  -4.6833590 -5.2680034 -6.8761868 -6.7822291 -2.8183381
## even    -0.6238642 -0.3893936 -1.0262648 -0.6709841 -0.7647598
## random   1.4987019  1.5789303  1.6582286  1.4879137  0.3567713
```

Now we can see an interesting thing: while the first three sets of SES values from null models were telling us a similar story, the results from the new null model differ particularly for "clump2a". This can be due to many reasons, such as the different indices of functional diversity that we are using (mpd is an indicator of functional divergence versus Faith's PD which is an indicator of functional richness) or the fact that in SES5 we are considering species abundances.

7.9 Examples with real dataset

Now that we have learned how to make some basic null models with very simple datasets, we can go a bit further and use our real dataset (remember, 67 annual plant assemblages across grazing and productivity gradients) to explore community assembly patterns.

7.9.1 Environmental filtering

We will start testing whether there is habitat filtering associated to grazing and productivity. Our hypothesis is low productivity and high grazing will restrict the set of traits that can be found in the communities. We will use the *independentswap* null model for this. As we have already seen, this null model works with presence-absence data better than with abundance data, which is already a limitation. The interesting thing is that it keeps constant both the species richness in each plot and the regional frequency of species. We will create a presence-absence matrix for our communities and then apply the function. Note that we will increase the number of randomizations to 500 in the example to make things a bit more "realistic", anyway, an even higher number is generally recommendable (for example, in Carmona et al., 2015a, we used 10,000 repetitions for this dataset). In each iteration, we will estimate the mpd diversity index, for which we first need to create a matrix of dissimilarity between pairs of species based on the traits (using the gower distance). The null model can be done with the `ses.mpd` function, from which we will extract the column containing the SES values:

```
dissTraits <- as.matrix(gowdis(spxtraits))
plotxspPA <- replace(plotxsp, plotxsp > 0, 1)
SESMPD <- ses.mpd(samp = plotxspPA, dissTraits, null.model = "independentswap",
                  runs = 500, abundance.weighted = F)
SESMPD <- SESMPD$mpd.obs.z
```

Now we have SES values showing how the observed mpd values in each site differ from those expected by chance if all species where able to live everywhere, while keeping constant the number of species in each site and the total number of occurrences of each species. We can use the environmental information contained in the `plotxenv` object to make a model to examine whether soil resource availability (shown in the column `Productivity`), grazing intensity (in the column `Grazing`), and their interaction affect trait diversity in the dataset:

```
model <- lm(SESMPD ~ Productivity * Grazing, data = plotxenv)
anova(model)
```

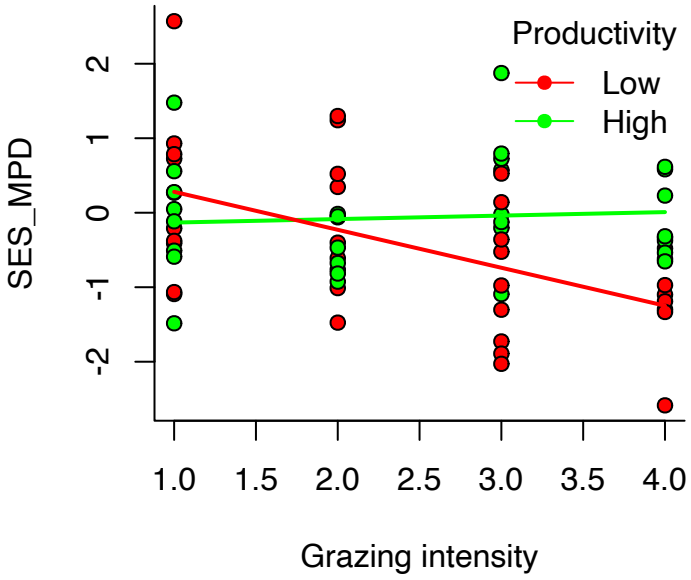
```
## Analysis of Variance Table
##
## Response: SESMPD
##                      Df Sum Sq Mean Sq F value   Pr(>F)
## Productivity          1  2.746  2.7456  3.8774 0.053341 .
## Grazing               1  4.570  4.5705  6.4545 0.013543 *
## Productivity:Grazing  1  6.552  6.5525  9.2536 0.003424 **
## Residuals            63 44.610  0.7081
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(model)
```

```
##
## Call:
## lm(formula = SESMPD ~ Productivity * Grazing, data = plotxenv)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.37183 -0.57672  0.01611  0.57484  2.29054
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.17895    0.35014  -0.511  0.61108
## ProductivityLow           0.96906    0.49517   1.957  0.05478 .
## Grazing                   0.04681    0.12956   0.361  0.71911
## ProductivityLow:Grazing -0.55644    0.18292  -3.042  0.00342 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8415 on 63 degrees of freedom
## Multiple R-squared:  0.2372, Adjusted R-squared:  0.2008
## F-statistic: 6.529 on 3 and 63 DF,  p-value: 0.0006473
```

It looks as if the interaction between grazing and productivity is actually the most important thing going on here. Before going deeper into statistical details, we can make a plot to show these results:

```
plot(SESMPD ~ plotxenv$Grazing, pch = 21,
     bg = ifelse(plotxenv$Productivity == "Low", "red", "green"),
     xlab = "Grazing intensity", ylab = "SES_MPD", bty = "l")
legend("topright", legend = c("Low", "High"), bty = "n", title = "Productivity",
       col = c("red", "green"), pch = 16, lwd = 1)
# Line for high productivity:
curve(coef(model)[1] + coef(model)[3] * x, add = T, col = "green", lwd = 2)
# Line for low productivity:
curve(coef(model)[1] + coef(model)[2] + (coef(model)[3] + coef(model)[4]) * x,
      add = T, col = "red", lwd = 2)
```



The results suggest that when productivity is high, grazing intensity does not affect much functional diversity (see the non-significant effect of Grazing in the model summary). However, the importance of grazing is much higher when productivity is low (see the Productivity:Grazing interaction). As a result, low productivity sites that are not grazed (grazing intensity = 1, left side of the figure) have higher functional diversity than high productivity sites that are not grazed. This is probably because productive sites with no grazing are mostly dominated by species that have specific traits that allow them to compete strongly for light and exclude species without those traits. On the other end of the grazing gradient (very high grazing pressure), the functional diversity of low productivity sites is much lower. We think that this is because there are very few species that can simultaneously cope with high grazing pressure and low productivity, and these species have very specific sets of traits, so that they are all very similar (e.g. very small plants).

7.9.2 Biotic interactions

In Carmona et al. (2015a) we were also interested in how biotic interactions changed across the environmental gradients considered. For this, we wanted to get rid of any effect of the environment as much as possible, and work only with a community that has been "filtered" already. The most strict way to do this is to consider, for each site, only the species that are present. The idea is that if there is limiting similarity occurring in the site, the most abundant species will tend to have different trait values (so that functional diversity will be higher than expected if the abundance of the species is random). We can test for this by randomizing the abundance of species within sites (but restricting this randomization only to species that are actually present in the site already! Note that this null model is hence very very restrictive!). The randomization here is a bit more complex, but the idea is similar. We will use again the `melodic` function to calculate mpd, because, as we showed in R material 5, the `mpd` function from `picante` does not work very well when we want to consider species abundances (it returns rao instead of mpd). At this point you should already have some understanding of the different steps involved in this process, so that we will do all estimations in a single chunk of code. However, we have added comments to the code so that everything is more or less easy to follow. You will also notice that as our randomizations are getting more complex, they take more time to process:

```
numberReps <- 500
# 1. Estimate observed values of mpd using the melodic function:
observedMPD <- melodic(samp = plotxsp, dis=dissTraits ,
                       type="abundance")$abundance$mpd
# 2. Prepare a matrix to store the results of each iteration of the null model:
```

```
simMPD <- matrix(NA, nrow = numberReps, ncol = nrow(plotxsp),
                 dimnames = list(paste0("Sim.", 1:numberReps), rownames(plotxsp)))
# 3. Make the null model for each site.
for (i in 1:ncol(simMPD)){ #For each site
  # 3.1. create a matrix full of zeroes to store the 500 null communities for the site i:
  randomComms <- matrix(0, nrow = numberReps, ncol = ncol(plotxsp),
                        dimnames = list(paste0("Sim.", 1:numberReps), colnames(plotxsp)))
  # 3.2 check which species are present at site i. Those are the ones whose abundances we will shu
  spInComm <- which(plotxsp[i,] > 0)
  realisedAbund <- plotxsp[i, spInComm]
  # 3.3. in each repetition of the null model, shuffle the aboundances of species present in site
  for(j in 1:numberReps){
    randomisedAbund <- as.numeric(sample(realisedAbund))
    randomComms[j,  spInComm] <- randomisedAbund
  }
  # 3.4. Estimate mpd for each repetition, and store the results in simMPD
  simMPD[, i] <- melodic(samp=randomComms, dis=dissTraits ,
                         type="abundance")$abundance$mpd
}
# 4. Estimate SES for each site
SESxSampleWithin <- (observedMPD - colMeans(simMPD)) / apply(simMPD, 2, sd, na.rm = T)
```

Good! Lets take a look at the effects of productivity and grazing:

```
model2 <- lm(SESxSampleWithin ~ Productivity * Grazing, data = plotxenv)
anova(model2)
```

```
## Analysis of Variance Table
##
## Response: SESxSampleWithin
##                      Df Sum Sq Mean Sq F value  Pr(>F)
## Productivity          1  1.690  1.6904  2.1827 0.14456
## Grazing               1  1.072  1.0719  1.3841 0.24383
## Productivity:Grazing  1  4.254  4.2544  5.4933 0.02226 *
## Residuals            63 48.791  0.7745
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

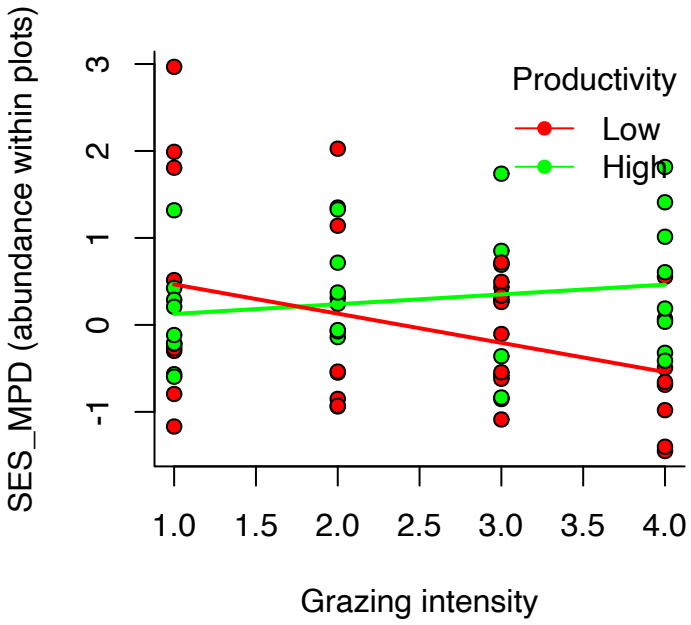
```
summary(model2)
```

```
##
## Call:
## lm(formula = SESxSampleWithin ~ Productivity * Grazing, data = plotxenv)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6342 -0.7040 -0.1141  0.5465  2.5015
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.01338    0.36618   0.037   0.9710
## ProductivityLow           0.78821    0.51785   1.522   0.1330
## Grazing                   0.11242    0.13550   0.830   0.4099
## ProductivityLow:Grazing -0.44836    0.19130  -2.344   0.0223 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.88 on 63 degrees of freedom
## Multiple R-squared:  0.1257, Adjusted R-squared:  0.0841
## F-statistic:  3.02 on 3 and 63 DF,  p-value: 0.03622
```

Again, the interaction between grazing and productivity seems to be the most important driver of variation. Let's plot the results again

```
plot(SESxSampleWithin ~ plotxenv$Grazing, pch = 21,
     bg = ifelse(plotxenv$Productivity == "Low", "red", "green"),
     xlab = "Grazing intensity", ylab = "SES_MPD (abundance within plots)",
     bty = "l")
legend("topright", legend = c("Low", "High"), bty = "n",
       title = "Productivity", col = c("red", "green"), pch = 16, lwd = 1)
# Line for high productivity:
curve(coef(model2)[1] + coef(model2)[3] * x, add = T, col = "green", lwd = 2)
# Line for low productivity:
curve(coef(model2)[1] + coef(model2)[2] + (coef(model2)[3] + coef(model2)[4]) * x,
      add = T, col = "red", lwd = 2)
```



Interesting! The results are apparently very similar to what we observed before, with a negative effect of grazing on diversity in low productivity sites and a rather neutral effect of grazing on high productivity sites. As a consequence, SES values in non-grazed areas do not differ much between productive and non-productive sites, but they differ much more in highly grazed areas. But note that the message is slightly different. What this null model results are showing as is that grazing in low productivity areas imposes a constrain on trait values that goes beyond the effect of environmental filtering, so that the most abundant species observed in low productivity conditions become more and more similar in their traits as grazing pressure increases.

7.9.3 The site specific species pool and dark diversity

As we mention in the reference book, the selection of a relevant pool of species for randomizations is one of the most important steps when designing a null model. There are many different pools that can be considered for a particular study. For example, we could be interested in examining the effect of grazing on productive sites, without considering

species that are only present in unproductive sites, or vice versa. In that case, we could repeat the *independentswap* model, but including in the randomization only the sites with the same productivity level. An interesting approach in this line is to consider the *site specific species pool*, which is composed by the set of species from the regional pool that can potentially thrive under the site's ecological conditions. The site specific species pool includes the species that are present in the site when we sampled it (we know that they can live there, because we have seen them!), plus other species that are absent in the moment we sampled. This second subset of species are what is called the *dark diversity* of the site (Pärtel et al., 2011). Using site specific species pools in our null models can help us to explore questions about why some species that are otherwise suitable are absent from a particular site: perhaps the traits of absent species suggest some particular mechanism, like low dispersal ability? The main problem with this approach is that dark diversity needs to be estimated as it is not something that we can measure through observation in the field. For this, the most commonly used approaches are based on species co-occurrences. Imagine that species A and species B co-occur very often, whereas species C does not co-occur with them because it has different ecological requirements. If we sample a site, and we find species B, then we will tend to think that species A is part of the dark diversity of the site, but species C is not. If we repeat this exercise for all absent species (i.e. all species in our dataset that are not present in the considered site), we can get an estimation of whether each of these species belong to the dark diversity of the site. Methods to do this are out of the scope of this material, but some of them are implemented in the `DarkDiv` package. Let's estimate the site specific species pools of our sites using the method "ThresholdBeals", which is a method that is based on the above mentioned observed and expected co-occurrences of species:

```
sitePools <- DarkDiv(x = plotxspPA, method = "ThresholdBeals", limit = "min", const = 0.01)
```

The `sitePools` object contains different things. We can first focus on `sitePools$indication`, which is the indication matrix between pairs of species. Within this matrix, columns are for indicator species and rows for target species, and the value of the cell tell us what is the probability that the target species is in the dark diversity of the site given that the indicator species has been sampled:

For example, if we go to a site in which *Agrostis castellana* is present, it is very unlikely that *Aira cupaniana* is part of the dark diversity of that site. Species that are more frequent in the dataset tend to have higher probabilities when applying this method, which is not particularly desirable, and for which researchers usually apply some abundance threshold to transform these probabilities into a binary value according to which a species is part (1) or not (0) of the dark diversity of a site. This is contained in `sitePools$Pool`, which includes the site specific species pool of each site, in a matrix that has the same structure as the `plotxspPA` matrix:

We can now make a null model in which we randomly sample as many species as we observed in each site from the site specific species pool. We will use this null model to examine if there are differences between the average trait values of the species that we observe in the sites (*observed diversity*) and what we would expect if all species from the site specific species pool had equal chance to be present:

```
numberReps <- 500
# 1. Estimate observed values of mpd using the melodic function:
observedCWM <- functcomp(x = spxtraits, a = as.matrix(plotxspPA))
# 2. Prepare a matrix to store the results of each iteration of the null model, for each trait:
simHeight <- simSLA <- simSeed <- matrix(NA, nrow = numberReps,
                                         ncol = nrow(plotxspPA),
                                         dimnames = list(paste0("Sim.", 1:numberReps),
                                                         rownames(plotxspPA)))
# 3. Make the null model for each repetition:
for(rep in 1:numberReps){
  plotxspPAAux <- plotxspPA
  plotxspPAAux[,] <- 0
```

```
  for (i in 1:nrow(plotxspPAAux)) { #for each site
    richnessSite <- rowSums(plotxspPA)[i]
    siteSpecificPool <- sitePools$Pool[i, ]
    spInSite <- which(siteSpecificPool == 1)
    spInNull <- sample(spInSite, size = richnessSite)
    plotxspPAAux[i, spInNull] <- 1
  }
  simCWM <- functcomp(x = spxtraits, a = as.matrix(plotxspPAAux))
  simHeight[rep,] <- simCWM$Height
  simSLA[rep,] <- simCWM$SLA
  simSeed[rep,] <- simCWM$SeedMass
}
SESxSitePoolHeight<-(observedCWM$Height - colMeans(simHeight)) / apply(simHeight, 2, sd, na.
SESxSitePoolSLA<-(observedCWM$SLA - colMeans(simSLA)) / apply(simSLA, 2, sd, na.rm=T)
SESxSitePoolSeed<-(observedCWM$SeedMass - colMeans(simSeed)) / apply(simSeed, 2, sd, na.rm=T
```

Great. Let's finally check the effects of productivity and grazing:

```
modelHeight <- lm(SESxSitePoolHeight ~ Productivity * Grazing, data = plotxenv)
anova(modelHeight)
```

```
## Analysis of Variance Table
##
## Response: SESxSitePoolHeight
##                      Df Sum Sq Mean Sq F value   Pr(>F)
## Productivity          1  1.492  1.4916  1.5388 0.219389
## Grazing               1  9.277  9.2770  9.5708 0.002946 **
## Productivity:Grazing  1  0.045  0.0447  0.0461 0.830643
## Residuals            63 61.066  0.9693
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(modelHeight)
```

```
##
## Call:
## lm(formula = SESxSitePoolHeight ~ Productivity * Grazing, data = plotxenv)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.86463 -0.68915  0.03834  0.68128  2.20776
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.95616    0.40966   2.334   0.0228 *
## ProductivityLow           0.19057    0.57934   0.329   0.7433
## Grazing                  -0.35410    0.15159  -2.336   0.0227 *
## ProductivityLow:Grazing   0.04596    0.21401   0.215   0.8306
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9845 on 63 degrees of freedom
## Multiple R-squared:  0.1504, Adjusted R-squared:   0.11
## F-statistic: 3.719 on 3 and 63 DF,  p-value: 0.01581
```

```r
modelSLA <- lm(SESxSitePoolSLA ~ Productivity * Grazing, data = plotxenv)
anova(modelSLA)
```

```
## Analysis of Variance Table
##
## Response: SESxSitePoolSLA
##                      Df Sum Sq Mean Sq F value  Pr(>F)
## Productivity          1  1.192  1.1921  1.2439 0.26895
## Grazing               1  4.990  4.9904  5.2072 0.02588 *
## Productivity:Grazing  1  0.001  0.0006  0.0006 0.97995
## Residuals            63 60.377  0.9584
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(modelSLA)
```

```
##
## Call:
## lm(formula = SESxSitePoolSLA ~ Productivity * Grazing, data = plotxenv)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.12289 -0.60139 -0.05223  0.50693  2.72618
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.91148    0.40734  -2.238   0.0288 *
## ProductivityLow          -0.25748    0.57606  -0.447   0.6564
## Grazing                   0.24549    0.15073   1.629   0.1084
## ProductivityLow:Grazing -0.00537    0.21280  -0.025   0.9799
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.979 on 63 degrees of freedom
## Multiple R-squared:  0.0929, Adjusted R-squared:  0.0497
## F-statistic: 2.151 on 3 and 63 DF,  p-value: 0.1027
```

```r
modelSeed <- lm(SESxSitePoolSeed ~ Productivity * Grazing, data = plotxenv)
anova(modelSeed)
```

```
## Analysis of Variance Table
##
## Response: SESxSitePoolSeed
##                      Df Sum Sq Mean Sq F value Pr(>F)
## Productivity          1  0.215 0.21535  0.2156 0.6440
## Grazing               1  1.946 1.94590  1.9482 0.1677
## Productivity:Grazing  1  1.378 1.37751  1.3792 0.2447
## Residuals            63 62.925 0.99880
```

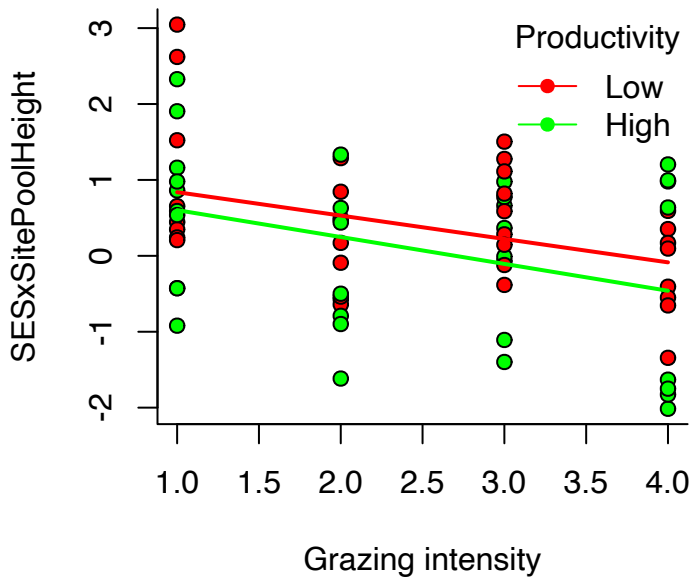
```r
summary(modelSeed)
```

```
##
## Call:
## lm(formula = SESxSitePoolSeed ~ Productivity * Grazing, data = plotxenv)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8589 -0.8106 -0.1000  0.7443  2.0812
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -0.26182    0.41584  -0.630    0.531
## ProductivityLow            0.51730    0.58809   0.880    0.382
## Grazing                   -0.02361    0.15388  -0.153    0.879
## ProductivityLow:Grazing -0.25513    0.21725  -1.174    0.245
##
## Residual standard error: 0.9994 on 63 degrees of freedom
## Multiple R-squared:  0.05324,    Adjusted R-squared:  0.00816
## F-statistic: 1.181 on 3 and 63 DF,  p-value: 0.3241
```

Looks like something is going on with height: grazing has a significant effect on plant height, so that the observed average height in grazed sites is smaller than what we would expect if the species that we find were a random subset of the site-specific species pool. This means that grazing is selecting for shorter species from the local species pool, and points to the prevalence of a strategy called "avoidance", in which plants avoid being eaten by grazers by being small.

```
plot(SESxSitePoolHeight ~ plotxenv$Grazing, pch = 21,
     bg = ifelse(plotxenv$Productivity == "Low", "red", "green"),
     xlab = "Grazing intensity", ylab = "SESxSitePoolHeight", bty = "l")
legend("topright", legend = c("Low", "High"), bty = "n", title = "Productivity",
       col = c("red", "green"), pch = 16, lwd = 1)
# Line for high productivity:
curve(coef(modelHeight)[1] + coef(modelHeight)[3] * x, add = T,
      col = "green", lwd = 2)
# Line for low productivity:
curve(coef(modelHeight)[1] + coef(modelHeight)[2] +
      (coef(modelHeight)[3] + coef(modelHeight)[4]) * x, add = T,
      col = "red", lwd = 2)
```



As we saw in this R material, there are many different ways to make null-models, and we hope we covered the different types existing and their technicalities.

8 – Traits and phylogeny

In this practical we will look at various ways of analyzing traits in combination with information on the phylogenetic relatedness between species. In most cases, this information will be available in the form of a phylogenetic tree. As this type of information contained in a phylogenetic tree is peculiar compared to the other types of data that we have met so far (i.e. mostly community, trait, and environmental data), and because this requires a different structure of storing this information in an R object, we will first turn our attention to importing and handling phylogenetic trees in R. Before we can do so, we need to load a number of packages (and install these in case they are not part of your R library yet).

8.1 Data

We will make use of various data sets throughout this R material. Because they are all quite specific to the different tasks, and the purposes quite different, we will read them in and explain them in the individual sections rather than doing that here.

8.2 Required packages and additional functions

We will use a selection of packages that are tailored to handle phylogenetic data and their analysis. None of them is part of base R, so please install them if you haven't already.

```
library(ape)
library(picante)
library(phytools)
library(Rphylopars)
```

We also need one package that is not available on CRAN but on github, so we need to install it from there with the help of the `devtools` package and its `install_github` function. (Install first `devtools` from CRAN if you haven't done yet.)

```
# devtools::install_github("jinyizju/V.PhyloMaker")
library(V.PhyloMaker)
```

8.3 Importing, handling, and visualizing phylogenetic trees

The most common form to encounter phylogenetic tree data is the so-called *Newick* format. It stores the topological characteristics of a phylogenetic tree with the help of brackets. Imagine the simplest case of a phylogenetic tree, with only one ancestor, which has two daughter taxa. Using the Newick format, we can represent such a tree by

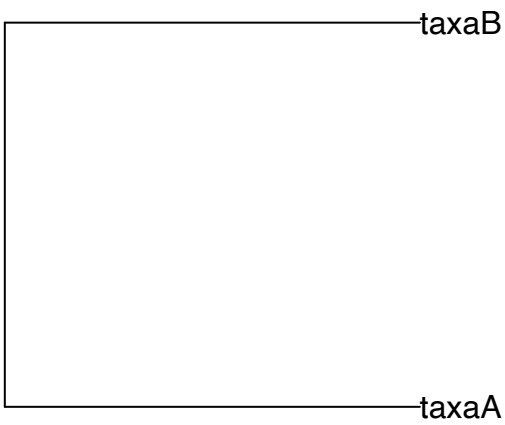
(taxaA, taxaB);

The semicolon at the end is expected by the Newick format, no matter how small or large the tree is. We can store this very simple tree as a character vector, and use the read.tree function to translate the Newick format into the `ape` way to store phylogenetic trees - as a `phylo` object.

```
tree_newick <- "(taxaA, taxaB);"
tree <- read.tree(text = tree_newick)
```

Before we have a look at how the structure of a `phylo` object looks like, we will plot the tree we just created. This helps us to more easily understand the additional modifications we will be making to our tree.

```
plot(tree)
```



This is the simplest tree we can get, showing us the two species, each sitting on their own branch.

Let's now see how the tree is actually stored in ape's own phylo object structure.

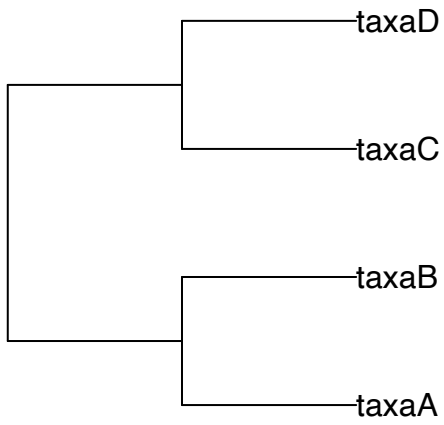
```
str(tree)
```

```
## List of 3
##  $ edge     : int [1:2, 1:2] 3 3 1 2
##  $ Nnode    : int 1
##  $ tip.label: chr [1:2] "taxaA" "taxaB"
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

The phylo object is a list, which in this case only has three elements. A matrix called `edge`, which contains the information how tips (i.e. species) and nodes are connected in the topology of the tree. `Nnode`, containing only one number, the number of nodes in the tree. In this case, this number is 1, since there is only a single node which connects taxaA and taxaB. Then there is a character vector called `tip.label`, which provides the names of the tips, i.e. the names of the species of the tree. In addition, the phylo object has to attributes, one which tells us that the object has the class phylo, and one that tells us that the tree is ordered cladewise.

In many cases, you will find two additional elements in a phylo object, which are not part of our very simple example tree. We will build a slightly more complex tree with four species to demonstrate this, and plot it right away.

```
tree_newick <- "((taxaA:1, taxaB:1)nodeA:1, (taxaC:1, taxaD:1)nodeB:1)nodeX;"
tree <- read.tree(text = tree_newick, show.node.label = TRUE)
plot(tree)
```



The tree now has two pairs of species that are sister taxa, connected by a common node (nodeX), which is at the same time the root of our tree (i.e. representing the oldest common ancestor in the tree; see Fig. 8.1. in the reference book). From this root, two other nodes descend (nodeA, nodeB). Each of these has two descendants, which make the four species (tips) of the tree. In the Newick code, behind each species and each node, we added a colon and a number. This number is the edge length (also called branch length), which connects a node or species to its ancestor. Here, for simplicity, we set all edges to a length of 1, and so all the branches appear with equal length in the plotted tree.

These additional data provided within the Newick format are now also reflected in the `phylo` object. In addition to the three elements explained above, two additional elements that represent the branch lengths (`edge.length`) and the node name (node.label‘) are added.

```
str(tree)
```

```
## List of 5
##  $ edge       : int [1:6, 1:2] 5 6 6 5 7 7 6 1 2 7 ...
##  $ edge.length: num [1:6] 1 1 1 1 1 1
##  $ Nnode      : int 3
##  $ node.label : chr [1:3] "nodeX" "nodeA" "nodeB"
##  $ tip.label  : chr [1:4] "taxaA" "taxaB" "taxaC" "taxaD"
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

Luckily, you don’t have to create the Newick format file for a tree by hand; imagine this if there are dozens or even hundreds of species in a tangled web of clustered brackets. Instead, if a given tree is available in this format, you can simply read it in from wherever the Newick file (usually with the file extension *.tre* or *.new*) is stored.

To read in the Newick format file that we will use for subsequent analyses with the same `read.tree` function, we specify a path on your computer from where this file should be read in. Make sure that the R working directory is set to the right folder on your computer. Note that other tree formats can also be used sometimes, which have their own specific read functions in the ape package.

```
tree_daphne <- read.tree(file = here::here("data", "chapter8", "DaPhnE_01.tre"))
str(tree_daphne)
```

```
## List of 6
##  $ edge       : int [1:8466, 1:2] 5123 5124 5125 5125 5126 5126 5127 5128 5128 5127 ...
##  $ edge.length: num [1:8466] 41.6 32.9 351 38 313 ...
##  $ Nnode      : int 3345
##  $ node.label : chr [1:3345] "Vascular_plants" "Lycopodiophyta" "Lycopodiales" "N4" ...
##  $ tip.label  : chr [1:5122] "Huperzia_selago" "Lycopodiella_inundata" "Lycopodium_clavat
```

```
##  $ root.edge  : num 1
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

There are quite some species in this tree: 5122 to be precise! The node names in some cases refer to ancestral nodes of entire taxonomic groups. For instance, "Lycopodiales" is the node from which all so-called club mosses descent, the oldest clade of vascular plants. This particular phylogeny, named DaPhnE, is an example of what is known as an informal supertree, and was put together and made available by Durka & Michalski (2012). It contains mainly species that are distributed in Central Europe. Even bigger super trees are currently available (Smith and Brown, 2018; Zanne et al., 2013), but since we are going to work with European plant species in our exercises, we are happy to use the Daphne tree for our purposes.

Nevertheless, we will demonstrate here shortly also the use of the recently published `V.phylomaker` package, which makes the phylogenetic supertree published by Smith & Brown (2018) easily accessible through R, and allows you to match your own species against this phylogeny. It also can draft (i.e. add) species onto the tree that are in your species list, but not in the published supertree.

The advantage of using `V.PhyloMaker` is that even species that are not in the backbone tree (which comes together with the package) are matched against the user-provided species list, provided that information on the genera and/or family names are present. `V.PhyloMaker` does this based on three different scenarios with regards to where exactly to draft those species on to the tree. We will not describe each of these here in detail but refer you to the article where these are described (Qian and Jin, 2016). To make `V.PhyloMaker` work, we need not only the species names, but also the names of genera and families with each of the species names, arranged in a data frame. For demonstration, we continue with a very small example species list.

```
species <- c("Abies alba", "Pinus nigra", "Cocos nucifera")
genus <- c("Abies", "Pinus", "Cocos")
family <- c("Pinaceae", "Pinaceae", "Arecaceae")
taxa <- data.frame(species = species,
             genus = genus,
             family = family)
tree_phylomaker <- V.PhyloMaker::phylo.maker(taxa)
```

```
## [1] "All species in sp.list are present on tree."
```

Though we only use three species here, the process takes a bit, because these three species are being matched against a huge phylogenetic backbone tree that contains more than 70,000 species! We also get some feedback from the function that tells us that all the species that are in our species list are actually present in the backbone tree, so none of the mentioned scenarios need to be used to draft them on the tree. To see what the output actually contains, we use the `str` function.

```
str(tree_phylomaker)
```

```
## List of 2
##  $ scenario.3  :List of 5
##   ..$ edge       : int [1:4, 1:2] 4 4 5 5 1 5 2 3
##   ..$ edge.length: num [1:4] 325.1 283.4 41.7 41.7
##   ..$ Nnode      : int 2
##   ..$ node.label : chr [1:2] "Spermatophyta" ""
##   ..$ tip.label  : chr [1:3] "Cocos_nucifera" "Pinus_nigra" "Abies_alba"
##   ..- attr(*, "class")= chr "phylo"
```

```
##   ..- attr(*, "order")= chr "cladewise"
##  $ species.list:'data.frame':    3 obs. of  4 variables:
##   ..$ species: chr [1:3] "Abies alba" "Pinus nigra" "Cocos nucifera"
##   ..$ genus  : chr [1:3] "Abies" "Pinus" "Cocos"
##   ..$ family : chr [1:3] "Pinaceae" "Pinaceae" "Arecaceae"
##   ..$ status : chr [1:3] "prune" "prune" "prune"
```

There is only a result for scenario 3, which is the default used by the function. In this case it is anyway not effective, as all three species are present in the backbone tree. We can also see this in the second element of the output object called `species.list`, which is a copy of the data frame of species, genus, and family names that we provided, with an additional column called `status`, which informs us that all three species have been "pruned" from the backbone tree. The first element, `scenario.3` contains the phylogenetic tree of the three species, in form of the now already familiar `phylo` object.

8.4 Estimating the phylogenetic signal of a trait

To demonstrate how to calculate the phylogenetic signal of a given trait, we also need to import some trait data. We use here a subset of traits available from the LEDA data base. LEDA is a data base of mostly North-West European plant species that can be accessed as .txt files on the webpage. We select traits from the so-called LHS scheme: specific leaf area (SLA), plant height, and seed mass (see Chapter 3 in the reference book). We are not using the raw data here, but we already aggregated the data at the species level, and omitted additional information (like geographic reference, number of replicates) we don't need. (Should you be interested in using this data for your own research you can, but you should start from the raw data that is downloadable from the website. See also R material Ch 2. Do not use the pre-aggregated data used here for your own purpose.)

```
lhs <- read.table(here::here("data", "chapter8", "lhs.txt"), header = TRUE)
str(lhs)
```

```
## 'data.frame':    2560 obs. of  6 variables:
##  $ sla       : num  2.65 2.88 3.21 3.25 3.65 ...
##  $ height    : num  0.65 0.95 0.183 0.225 0.45 ...
##  $ sm        : num  0.0353 NA 0.5434 NA 0.0754 ...
##  $ log_sla   : num  0.975 1.058 1.168 1.179 1.293 ...
##  $ log_height: num  -0.4308 -0.0513 -1.6964 -1.4917 -0.7985 ...
##  $ log_sm    : num  -3.34 NA -0.61 NA -2.59 ...
```

The phylogenetic tree contains more and partly different species than the species trait data, so the tree is pruned first to contain only species that are also in the trait data set. We encounter a very common hick-up when trying to match trait and phylogeny data (or in fact any other kind of data), and that is different naming standards. In this case, spaces in species names in the tree are substituted with underscores ("_"), whereas in the trait data they are not. (See also R material Ch 2 for another common problem, when different taxonomic concepts are used across the different data.)

```
head(tree_daphne$tip.label)
```

```
## [1] "Huperzia_selago"           "Lycopodiella_inundata"
## [3] "Lycopodium_clavatum"       "Lycopodium_annotinum"
## [5] "Diphasiastrum_zeilleri"    "Diphasiastrum_tristachyum"
```

```
head(rownames(lhs))
```

```
## [1] "Juncus maritimus"       "Equisetum hyemale"      "Salicornia europaea"
## [4] "Festuca psammophila"    "Juncus inflexus"        "Loiseleuria procumbens"
```

This problem can be solved by changing the names of either, according to the standard in the other. To get rid of the underscores in the species names of the phylogenetic tree:

```
tree_daphne$tip.label <- gsub("_", " ", tree_daphne$tip.label)
```

Then, the trait and phylogenetic tree data can be matched by their species names. To make life easier later on, all NAs that occur in the trait data are omitted. This is a bit of a rough measure, used here for the sake of demonstrating the different methods. For more meaningful ways to tackle issues with NA values, see R material Ch 2. The `na.omit` function has omitted all rows from the data frame that contain an NA value for any of the traits. The `match_data` function will match the data in the phylogenetic tree with the data in the trait data frame, on the basis of equal species names in both datasets. This function also orders the species in the trait data according to the order of the species along the tips of the phylogenetic tree, as many functions working with such data require the same order based on species across trait and tree data. The output of `match_data` is a list with three elements, which hold the data after matching by species names of community data, trait data, and phylogenetic tree data, respectively. As only trait data are matched against the phylogenetic tree, the first element is empty in this case. We can call the elements of the object returned by `match_data` by their names with `$traits` and `$tree`.

```
source(here::here("data", "chapter8", "match_data.R"))
match_traits_phylo <- match_data(traits = na.omit(lhs), tree = tree_daphne)
str(match_traits_phylo)
```

```
## List of 3
##  $ com   : NULL
##  $ traits:'data.frame':  1362 obs. of  6 variables:
##   ..$ sla       : num [1:1362] 13.85 6.84 5.35 6.85 10.19 ...
##   ..$ height    : num [1:1362] 0.125 40 34.5 50 35 50 17.5 0.03 0.03 0.4 ...
##   ..$ sm        : num [1:1362] 0.0001 23.9 7.572 6.6637 6.24 ...
##   ..$ log_sla   : num [1:1362] 2.63 1.92 1.68 1.92 2.32 ...
##   ..$ log_height: num [1:1362] -2.08 3.69 3.54 3.91 3.56 ...
##   ..$ log_sm    : num [1:1362] -9.21 3.17 2.02 1.9 1.83 ...
##   ..- attr(*, "na.action")= 'omit' Named int [1:1195] 2 4 17 23 28 30 48 51 57 60 ...
##   .. ..- attr(*, "names")= chr [1:1195] "Equisetum hyemale" "Festuca psammophila" "Equisetum va
##  $ tree  :List of 5
##   ..$ edge       : int [1:2558, 1:2] 1363 1363 1364 1365 1366 1367 1368 1369 1369 1368 ...
##   ..$ edge.length: num [1:2558] 425.5 70.5 69 85 25 ...
##   ..$ Nnode      : int 1197
##   ..$ node.label : chr [1:1197] "Vascular_plants" "Spermatophyta" "Pinales" "Pinaceae" ...
##   ..$ tip.label  : chr [1:1362] "Huperzia selago" "Pinus nigra" "Pinus sylvestris" "Picea abies
##   ..- attr(*, "class")= chr "phylo"
##   ..- attr(*, "order")= chr "cladewise"
```

```
head(match_traits_phylo$traits)
```

```
##                       sla height        sm  log_sla log_height    log_sm
## Huperzia selago  13.850000  0.125  0.000100 2.628285  -2.079442 -9.210340
## Pinus nigra       6.838954 40.000 23.900000 1.922635   3.688879  3.173878
## Pinus sylvestris  5.347474 34.500  7.572000 1.676624   3.540959  2.024457
```

```
## Picea abies       6.846285 50.000  6.663667 1.923706   3.912023  1.896670
## Larix decidua    10.192480 35.000  6.240000 2.321650   3.555348  1.830980
## Abies alba        5.870000 50.000 79.200000 1.769855   3.912023  4.371976
```

```
match_traits_phylo$tree
```

```
##
## Phylogenetic tree with 1362 tips and 1197 internal nodes.
##
## Tip labels:
##   Huperzia selago, Pinus nigra, Pinus sylvestris, Picea abies, Larix decidua, Abies alba,
## Node labels:
##   Vascular_plants, Spermatophyta, Pinales, Pinaceae, N256, N257, ...
##
## Rooted; includes branch lengths.
```

If the data is processed and prepared in this way, the phylogenetic signals of the (log-transformed) traits can be assessed. We choose here two traits, seed mass and SLA. The most widely available and used methods to estimate phylogenetic signal are Blomberg's K (Blomberg et al., 2003), and Pagel's lambda (Pagel, 1999). See Chapter 8 in the reference book for more details.

```
# Extract single trait variables
sla <- as.matrix(match_traits_phylo$traits)[, 4]
sm <- as.matrix(match_traits_phylo$traits)[, 6]

# Estimate phylogenetic signal with function phylosig from package phytools
phylosig(match_traits_phylo$tree, sm, method = "K")
```

```
##
## Phylogenetic signal K : 0.194132
```

```
phylosig(match_traits_phylo$tree, sm, method = "lambda")
```

```
##
## Phylogenetic signal lambda : 0.987503
## logL(lambda) : -2289.47
```

```
phylosig(match_traits_phylo$tree, sla, method = "K")
```

```
##
## Phylogenetic signal K : 0.151787
```

```
phylosig(match_traits_phylo$tree, sla, method = "lambda")
```

```
##
## Phylogenetic signal lambda : 0.901098
## logL(lambda) : -674.501
```

The results from both indices suggests considerable phylogenetic signal in seed mass and SLA, though slightly weaker in the latter. Hence, evolutionary related species are more similar in seed mass and SLA than distantly related species. Note that there is a difference in the range of possible values for K and lambda. While lambda ranges from 0 (no phylogenetic signal) to 1 (very strong signal), Blomberg's K can have values larger than 1, in which case K indicates that the species' traits are even more similar among related species than one would expect (i.e. than under a Brownian motion model of evolution; see Chapter 8 of the reference book for more details).

8.5 Traitgrams

To illustrate the use of traitgrams, we will load some additional data to our R environment, this time plant community data of mesic grasslands in the Czech Republic. We load this data directly from an Rdata file called `mesicmeadows`, in which the actual community data are stored in the object mesicmeadows. It is a data frame with 296 samples (in rows) and 558 species (in columns).

```
load(here::here("data", "chapter8","mesicmeadows.Rdata"))
```

We select two distinct samples from the community data, with a high and low phylogenetic signal among the community members for the trait specific leaf area, respectively. Traitgrams are more meaningful for smaller numbers of species, e.g. of a single community, because a graph plotting a tree with several dozens or hundreds of species are difficult to visualize and interpret. As we only want to include in the traitgram the species that are actually occurring in the target community, we use the '!=0 indexing to retain only those species.

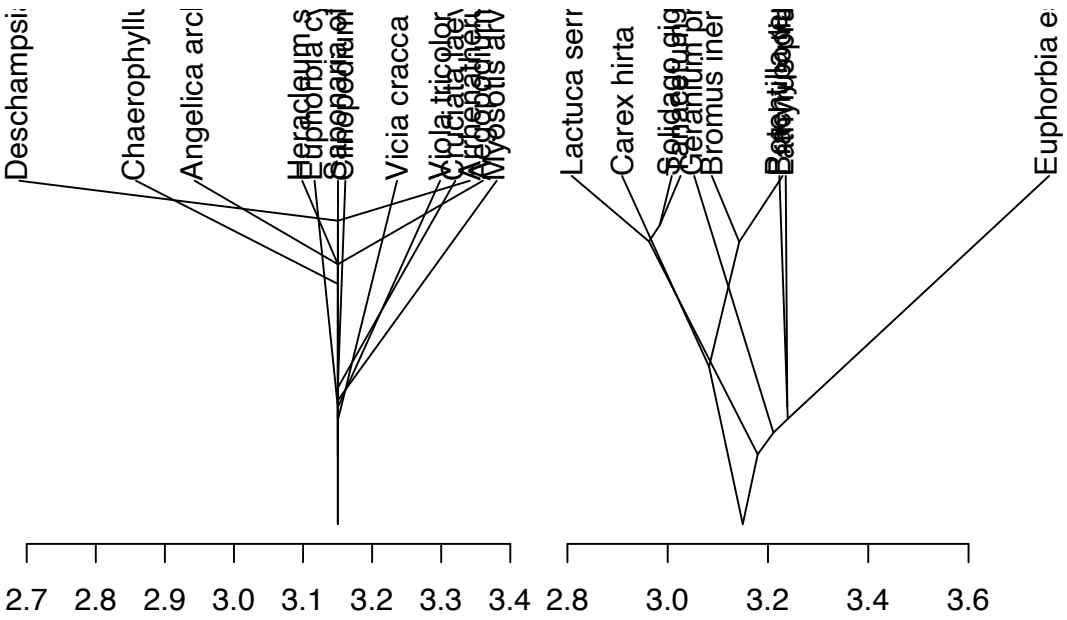
```
com1 <- mesicm[158,][, mesicm[158,] != 0]
com2 <- mesicm[21,][, mesicm[21,] != 0]
```

To plot the traitgrams for only the species occurring in those plots, the data again are matched with match_data, this time including also the plot data.

```
match1 <- match_data(com = com1, traits = na.omit(lhs), tree = tree_daphne)
match2 <- match_data(com = com2, traits = na.omit(lhs), tree = tree_daphne)
```

To produce the traitgrams the function `traitgram` from the picante package can be used. Both communities are displayed next to each other on the same graphic device. Note that we use the `multi2di` function to make the phylogenetic trees strictly dichotomous.

```
par(mfrow = c(1, 2))
traitgram(as.matrix(match1$traits["log_sla"]), multi2di(match1$tree),
          show.names = TRUE, cex = 0.7)
traitgram(as.matrix(match2$traits["log_sla"]), multi2di(match2$tree),
          show.names = TRUE)
```



The traitgrams nicely demonstrate the different phylogenetic signal in the communities. Essentially, traitgrams plot the topology of the phylogenetic tree of the species, but then use an axis of the trait values to "pull" the tips (with the species) to the position that displays their trait value. This might sound a bit complicated but it should become clearer when looking at the traitgrams. In the right-hand traitgram, species that are closely related also sit relatively close to each other on the trait axis, and as a result the traigram looks less "messy", with less intersecting branches.

```
phylosig(match1$tree, match1$traits[, "log_sla"])
```

```
## [1] "x has no names; assuming x is in the same order as tree$tip.label"

##
## Phylogenetic signal K : 0.259222
```

```
phylosig(match2$tree, match2$traits[, "log_sla"])
```

```
## [1] "x has no names; assuming x is in the same order as tree$tip.label"

##
## Phylogenetic signal K : 0.897076
```

Groups of species that display a low phylogenetic signal will always look like their tree structure is quite entangled, with a lot of branches crossing each other, like in the left traitgram. This is the result of closely related species having different trait values, and so their tips end up on distant point on the trait axis. For instance, on the left-hand traitgram, there are two grass species, i.e. of the genera Deschampsia and Arrentatherum that end up almost on opposite extremes of the trait axis.

8.6 Phylogenetic comparative methods

There are a multitude of methods available that account for phylogenetic dependence of species level data (see Chapter 8 in the reference book). As a simple example, let's look at one that has become known as "phylogenetic independent contrasts" (PIC). It is in fact one of the first methods that was proposed to account for phylogenetic relationships when correlating traits for a set of species, and became extremely popular.

To show how PIC are computed and analysed, we use seed mass and plant height. Since we are conducting a cross-species level analyses, community data are not needed, and we return to the `match_traits_phylo` object. It is always preferable to have a completely resolved (i.e. dichotomous) phylogeny, but often this is not the case. Hence the `multi2di` function is used to make it completely dichotomous. This function uses a trick to coerce the tree into a dichotomous structure, by adding branches of length 0. The `pic` function used to compute independent contrasts requires a dichotomous phylogeny. However, more advanced methods to account for phylogenetic relationships do not require completely dichotomous trees. After we treated the phylogeny to be dichotomous, we extract the single traits seed mass and height from the trait data frame, to be used in the `pic` function. Remember that the trait data have already been matched to the phylogenetic tree using the 'match_data' function.

```
tree_pic <- multi2di(match_traits_phylo$tree)
seedmass <- match_traits_phylo$trait[, "log_sm"]
height <- match_traits_phylo$trait[, "log_height"]
```

We use the `pic` function from package `ape` for calculating the independent contrasts.

```
pic_height <- pic(height, tree_pic)
pic_sm <- pic(seedmass, tree_pic)
```

The created objects `pic_height` and `pic_sm` contain nothing else but the calculated contrasts. We only display the first few not to overflow the screen.

```
head(pic_height)
```

```
## Vascular_plants   Spermatophyta         Pinales        Pinaceae            N256
##     -0.13751823      0.25789958      0.04299629     -0.01320389      0.01242393
##            N257
##     -0.01699354
```

Notice that there are no more species names, as the contrasts represent differences between trait values of species pairs (or pairs of nodes; see Chapter 8 in the reference book for more details). Hence, the names in `pic_height` are the names of nodes at which contrasts have been calculated. Some of these names are a cryptic combination of the letter N and a number, but there are also a few named ones, for nodes representing the common ancestor of a genus or a family.

The relationship between the PICs for height and seed mass can be analysed using a linear model, fitting the intercept through the origin (this is done by the -1 term in the model formula). To compare the result with a standard linear model without considering the phylogeny, we also run the linear model on the original seed mass and height values.

```
lm_pics <- lm(pic_sm ~ pic_height - 1)
lm_ord <- lm(seedmass ~ height)
```

Let's look at the summary output of these two models.

```
summary(lm_pics)
```

```
##
## Call:
## lm(formula = pic_sm ~ pic_height - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.06025 -0.14780  0.00282  0.14990  2.90280
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## pic_height  0.31114    0.03848   8.086 1.35e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3069 on 1360 degrees of freedom
## Multiple R-squared:  0.04587,    Adjusted R-squared:  0.04517
## F-statistic: 65.39 on 1 and 1360 DF,  p-value: 1.347e-15
```

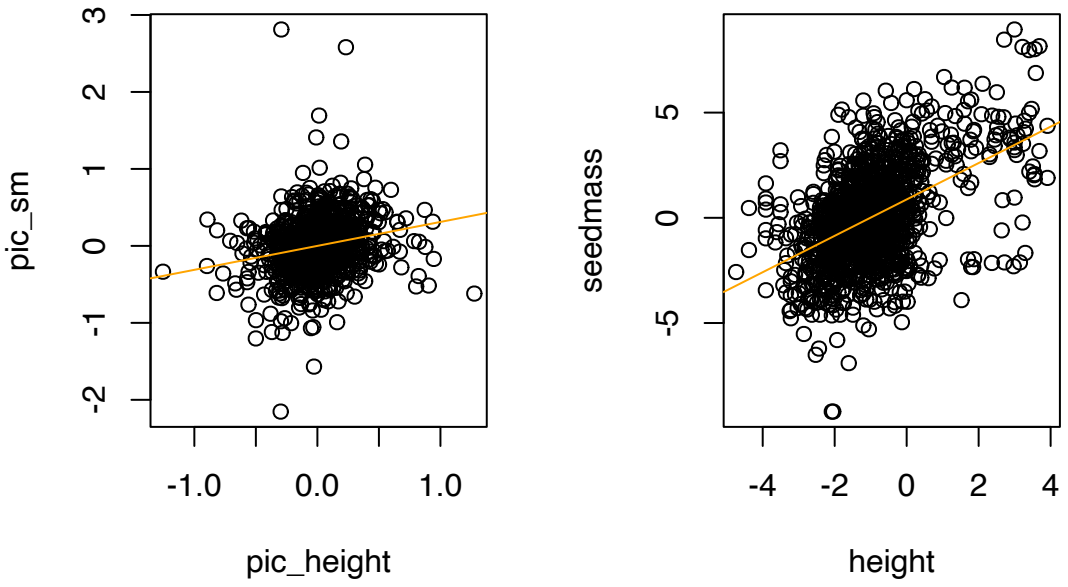
```
summary(lm_ord)
```

```
##
## Call:
## lm(formula = seedmass ~ height)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.319 -1.170  0.079  1.182  5.832
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.87270    0.06727   12.97   <2e-16 ***
## height       0.86479    0.04211   20.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.934 on 1360 degrees of freedom
## Multiple R-squared:  0.2367, Adjusted R-squared:  0.2362
## F-statistic: 421.8 on 1 and 1360 DF,  p-value: < 2.2e-16
```

Both models show a highly significant relationship between plant height and seed mass. Hence, higher plants do not only carry heavier seeds across extant species, but this relationship occurs throughout evolutionary times. In other words, a change in plant height during evolution, co-occurs with a change in seed mass in the same direction. To make this more clear, imagine two species A and B that have a common ancestor. For these two species we will calculate contrasts (i.e. differences) between the values of two given traits X and Y. Lets' say the contrasts for both traits of these two species are positive. If this example is extended to a more comprehensive phylogenetic tree with lots of those contrasts, and the *direction and magnitude of change* of the contrasts is always congruent for both traits, it implies that both traits did evolve in a correlated way.

```
par(mfrow = c(1, 2))
plot(pic_height, pic_sm)
abline(lm_pics, col = "orange")
plot(height, seedmass)
abline(lm_ord, col = "orange")
```



As is evident from the regression estimates and the slopes of the regression lines in the scatter plots, the slope from the standard regression is almost three times steeper than the one from the regression of PICs. This suggests that across the phylogenetic tree (and thus evolutionary time), co-occurring changes in the variables were less pronounced than what we can observe for extant species.

8.7 Imputing trait data with the help of phylogeny

The relationship between phylogenetic relatedness and trait similarity can also be used to obtain missing trait values for some species. If correlations between the different traits exist, as well as between the traits and the phylogeny (i.e. there is a phylogenetic signal), we can use these correlations to estimate missing trait values. Several R packages are available for data imputation, and we are going to focus here on `Rphylopars`, which allows a direct implementation of using phylogenetic information in the models for imputing the trait data. Other packages that do similar jobs are `mice` and `missForest`, though they require to first obtain so-called phylogenetic eigenvectors from a PCoA based on the phylogenetic distances between species.

```
library(Rphylopars)
?phylopars
```

The main function of this package comes with a long list of arguments, but in its most simple form a data frame with the trait data and a phylogenetic tree are needed. From these two data, we first need the entire data for our three traits, including the missing values (NA). Secondly, the species names must be explicit in the first column of the data frame and named `species`. This is because `phylopars` can even take into account intraspecific trait variability, and so the same species can appear in different rows of the data frame (which isn't possible when the species are row names of the data frame). We don't have such multiple species entries, so we are not going to use this functionality, but we need to specify the species names in the first column anyway.

```
match_pars <- match_data(traits = lhs[, 4:6], tree = tree_daphne)
head(match_pars$traits) # NAs are still there
```

```
##                           log_sla log_height   log_sm
## Huperzia selago          2.628285  -2.079442 -9.21034
## Lycopodiella inundata    4.080922  -3.218876       NA
## Lycopodium clavatum      3.081451  -2.162823       NA
## Lycopodium annotinum     3.218876  -1.897120       NA
## Selaginella selaginoides 3.198265  -2.590267       NA
## Selaginella helvetica          NA  -2.995732       NA
```

```
# combine the species names and trait data in single data frame with the species names as the firs
lhs_for_pars <- data.frame(species = rownames(match_pars$traits), match_pars$traits)
head(lhs_for_pars)
```

```
##                                       species  log_sla log_height   log_sm
## Huperzia selago                 Huperzia selago 2.628285  -2.079442 -9.21034
## Lycopodiella inundata     Lycopodiella inundata 4.080922  -3.218876       NA
## Lycopodium clavatum         Lycopodium clavatum 3.081451  -2.162823       NA
## Lycopodium annotinum       Lycopodium annotinum 3.218876  -1.897120       NA
## Selaginella selaginoides Selaginella selaginoides 3.198265  -2.590267       NA
## Selaginella helvetica       Selaginella helvetica       NA  -2.995732       NA
```

The `phylopars` function uses this data frame together with phylogeny. The `pheno_error` argument has to be set to `FALSE`, since intraspecific variability for estimating missing values is not used. By default, `phylopars` only returns the phylogenetic trait variance-covariance matrix. This matrix is used for estimations of the missing trait data, but is not very useful to us by itself. So the object created by the function is saved as a new object, from which the new trait data is extracted. The output we want is hidden under the cryptic name `anc_recon` (because in the background, as part of the estimation process, `phylopars` is reconstructing trait values for ancestral species).

```
gapfill_lhs <- phylopars(lhs_for_pars, match_pars$tree, pheno_error = FALSE)
head(gapfill_lhs$anc_recon)
```

```
##                            log_sla log_height    log_sm
## Huperzia selago           2.628285  -2.079442 -9.210340
## Lycopodiella inundata     4.080922  -3.218876 -3.421192
## Lycopodium clavatum       3.081451  -2.162823 -2.959643
## Lycopodium annotinum      3.218876  -1.897120 -2.908973
## Selaginella selaginoides 3.198265  -2.590267 -2.503722
## Selaginella helvetica     2.987733  -2.995732 -2.580907
```

8.8 Phylogenetic diversity

The evolutionary history of species contained in a phylogenetic tree can be expressed and interpreted by species pairwise distances. Every pair of species across a phylogenetic tree shares a common ancestor. For species that are closely related, this ancestor is not much evolutionary older than the species themselves. The phylogenetic distance between this pair of species is therefore relatively small. For two species that are taxonomically distant, one would have to go deep down the phylogenetic tree to find their last common ancestor, and the phylogenetic distance of this pair of species would be much larger. A distance matrix for all possible species pairs can be constructed, which has exactly the same data structure (i.e. a distance matrix) that is used in many approaches to calculate functional diversity (see R material Ch 5. Instead of characterizing how similar species in the same community are by their traits, their phylogenetic distances are used for the same purpose. Most of the indices used to calculate functional diversity are also applicable to estimate phylogenetic diversity (PD). The only technical step that is slightly different is the construction of the distance matrix from the phylogenetic tree. We use the `cophenetic` function from the `ape` package, which calculates phylogenetic distance as the sum of the branch lengths that separate two species. Since PD is calculated at the level of communities, the data that is returned includes the vegetation plot data of mesic meadows.

```
match_mm <- match_data(com = mesicm, tree = tree_daphne)
phylo_dist <- cophenetic(match_mm$tree)
```

The resulting matrix `phylo_dist` is based on a phylogenetic tree instead of traits. Note though, that the default output of the `cophenetic` function is a square matrix, not the triangular `dist` object that we encountered in R material Ch 3. The first few species can be extracted by

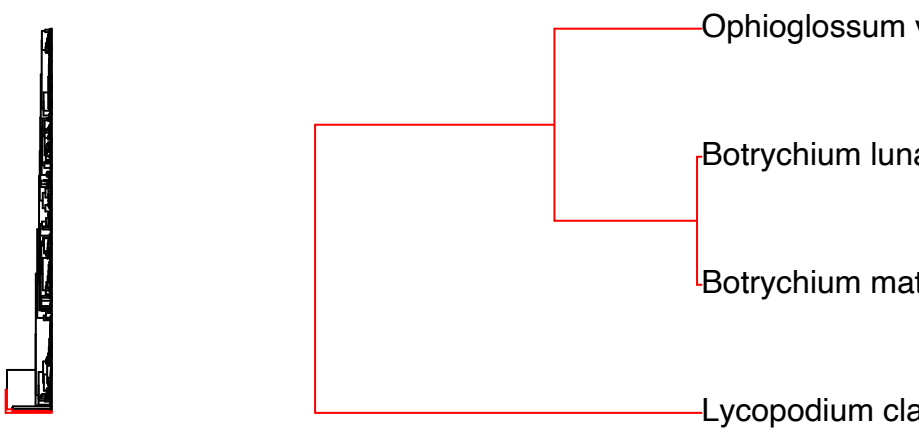
```
phylo_dist[1:4, 1:4]
```

```
##                            Lycopodium clavatum Botrychium matricariifolium
## Lycopodium clavatum                          0                          851
## Botrychium matricariifolium                851                            0
## Botrychium lunaria                         851                           10
## Ophioglossum vulgatum                      851                          324
##                            Botrychium lunaria Ophioglossum vulgatum
## Lycopodium clavatum                       851                    851
## Botrychium matricariifolium                10                    324
## Botrychium lunaria                          0                    324
## Ophioglossum vulgatum                     324                      0
```

To focus on this part of the tree, the `zoom` function (from package `ape`) can be used, if we indicate which species we want to zoom in on from the entire phylogenetic tree (the entire

tree is still shown as an overview on the left, though in our case there is not much to see because of its enormous size).

```
ape::zoom(match_mm$tree, 1:4)
```



All four species are fern-like species. As we can see in the square representation of the distance matrix, the distance of *Lycopodium clavatum* to itself is of course 0. The distance to all the other three displayed species is the same (851), as they are all separated from *L. clavatum* by the same distance. The two *Botrychium* species have the smallest distance between them (10), and since they are deviating from a common ancestor with the *Ophioglossum* species, they both have the same distance to *Ophioglossum*. Notice that these distances are exactly what is represented proportionally in the figure on the horizontal axis.

There are various packages and indices that can be used to calculate distance-based functional, and therefore also phylogenetic, diversity (see R material Ch 5. We show this with the mean pairwise distance index function `mpd`, but with the version from the melodic function. This function expects the distance matrix to be an actual matrix object, which is already the case for phylogenetic distances produced by `cophenetic`.

```
source(here::here("data", "chapter8","melodic.R"))
```

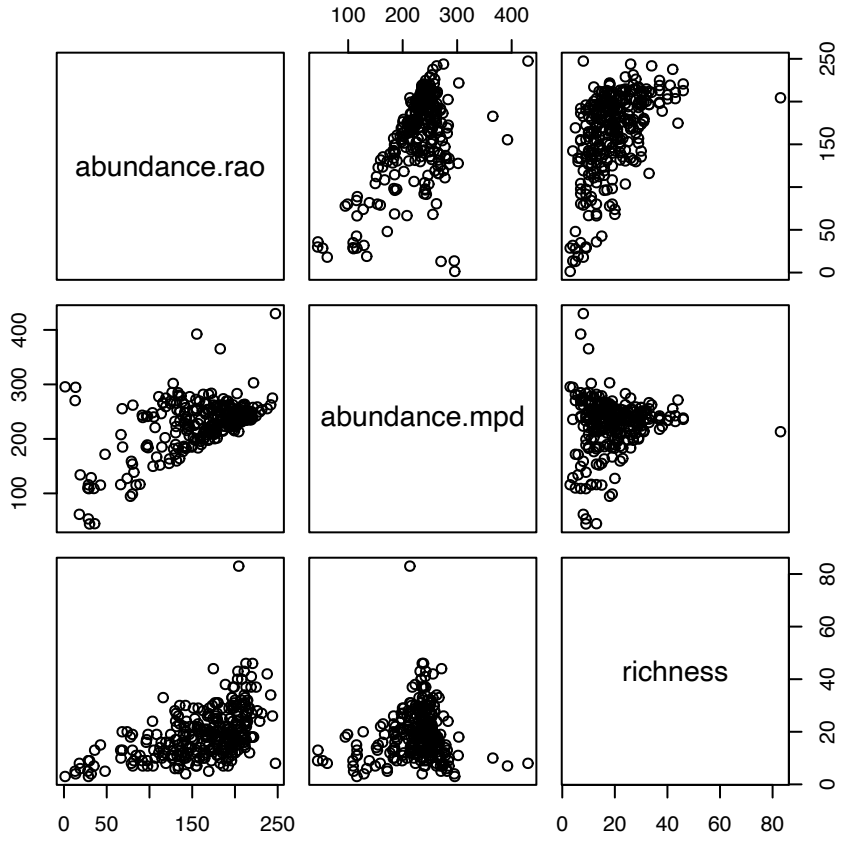
```
## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used

## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used

## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used

## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used

## Warning in if (class(samp) != "matrix") {: the condition has length > 1 and only
## the first element will be used

## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

```
melody <- melodic(match_mm$com, phylo_dist, type = "abundance")
```

```
## Warning in if (class(dis) != "matrix") {: the condition has length > 1 and only
## the first element will be used
```

The output from the `melodic` function is a list with four elements, which contain the following diversity indices: Simpson diversity, Rao's Q, mean pairwise distance (MPD), and species richness. Both Rao's Q and MPD, in the case of using phylogenetic distances instead of trait distances, are phylogenetic diversity measures (see R material Ch 5. They are both correlated with each other, while only Rao's Q is related to species richness (though mostly at low richness values).

```
pairs(as.data.frame(melody)[, 2:4])
```



8.9 How to combine trait and phylogenetic diversity

Different traits can be influenced by phylogenetic relatedness to various degrees, and this can depend on the particular sample of species under consideration. Despite the potential overlap between these two kinds of information (i.e. phylogenetic vs. trait differences among species), these two types of diversities (i.e. functional trait diversity and phylogenetic diversity) have initially been treated as separate entities in many studies, if they were considered together at all. More recent methodological approaches have tried to combine these two diversities into more complementary measures, and in the following code we will see how this can be done with the help of the 'decouple' function.

```
source(here::here("data", "chapter8","decouple.R"))
```

For the trait data, species that have missing trait information are removed. This is necessary for 'decouple' to work, because in the statistical steps behind the decoupling of functional and phylogenetic distances, the traits are used as response variables. As a matter of fact,

'decouple' would automatically exclude species with NA values for that trait. When working with your own data, you should also check data availability and see for how many and which species you are missing trait data, as it can greatly impact your results if you just "blindly" match phylogenetic tree and trait data against each other by their species names.

The decouple function only requires the trait and phylogenetic tree data, which is internally matched by species names (with the same `match_data` function we used above), so no need for us to do it manually as a first step this time. We use the fourth column, i.e. log-transformed SLA, from our lhs trait data frame to exemplify the use of the `decouple` function.

```
dcpld <- decouple(traits = na.omit(lhs[, 4, drop = FALSE]), tree = tree_daphne)
```

```
## Warning in is.euclid(x.dist): Zero distance(s)

## Warning in is.euclid(x.dist): Zero distance(s)
```

The `decouple` function returns a list of distance matrices (stored as square matrices, not as class distance). The name of each list element indicates what distance matrix we are looking at.

```
lapply(dcpld, class)
```

```
## $Pdist
## [1] "matrix" "array"
##
## $Fdist
## [1] "matrix" "array"
##
## $dcPdist
## [1] "matrix" "array"
##
## $dcFdist
## [1] "matrix" "array"
##
## $jointFPdist
## [1] "matrix" "array"
```

The first two elements contain "standard" species distances based on the phylogenetic and functional distances reflecting the phylogenetic tree and trait values, respectively. Then follow distance matrices for the decoupled phylogenetic and decoupled functional distances. The fifth element contains the joint distances. See Chapter 8 in the reference book for more information about these distance measures.

9 – Linking traits to ecosystem functions

In this exercise we will learn how to relate community traits to one or multiple *ecosystem functions*. This exercise follows *Chapter 9* of the reference textbook, so all theoretical issues beyond the approaches described in this exercise can be found there. We will first work with data used in Hanisch et al. (2020), to produce the same Fig. 9.3 as in the reference textbook. Then we will work with part of the data from a biodiversity experiment, with sown plant communities characterized by different levels of functional and phylogenetic diversity, from a temperate grassland and assess (or link them) to some functions of the ecosystem. These types of experiments are usually called Biodiversity-Ecosystem-Functions experiments, in short *BEFs*.

9.1 Data

We will work with data used in Hanisch et al. (2020) provided in the supplementary material of this interesting publication In the study the authors review evidences from the literature of the effect of traits on multiple ecosystem functions, expanding the original work by de Bello et al. (2010b) . The data used ('Hanischdata.txt') is a table in which, for each plant trait assessed the relationship with a number of ecosystem services is provided.

```
multiEFs <- read.delim(here::here("data", "chapter9", "Hanischdata.txt"), row.names = 1)
head(multiEFs)
```

```
##                                 Biomass.Production Soil.Fertility
## Leaf nitrogen content (LNC)                   0.50           0.42
## Plant height (VPH)                            0.50           0.66
## Specific leaf area (SLA)                      0.57          -0.06
## Leaf dry matter content (LDMC)               -0.50           0.17
## Plant C/N                                     0.00          -1.00
## Root depth                                    0.00           1.00
##                                 Erosion.Control Climate.Regulation
## Leaf nitrogen content (LNC)               -0.50                0.50
## Plant height (VPH)                         0.00                0.50
## Specific leaf area (SLA)                  -0.50               -0.25
## Leaf dry matter content (LDMC)             0.33                0.80
## Plant C/N                                  0.00                1.00
## Root depth                                -1.00               -1.00
##                                 Fodder.Quality Aesthetic.Appeal Water.Regulation
## Leaf nitrogen content (LNC)               0.5            -0.33                0
## Plant height (VPH)                       -0.5            -0.75               -1
## Specific leaf area (SLA)                  1.0             1.00               -1
## Leaf dry matter content (LDMC)           -1.0            -1.00                0
## Plant C/N                                 1.0             0.00                0
## Root depth                                0.0             0.00                1
##                                 Pollination Recreation.Heritage Biocontrol
## Leaf nitrogen content (LNC)               0                   -1      -0.67
## Plant height (VPH)                        0                   -1       0.00
## Specific leaf area (SLA)                  0                    0       0.00
## Leaf dry matter content (LDMC)            0                    0       0.00
## Plant C/N                                 1                    0       0.00
```

```
## Root depth                                0                   0        0.00
##                                 Water.Purification
## Leaf nitrogen content (LNC)                     -1
## Plant height (VPH)                               0
## Specific leaf area (SLA)                        -1
## Leaf dry matter content (LDMC)                   0
## Plant C/N                                        0
## Root depth                                       0
```

The value (ranging from -1 to +1) in each 'cell' of the matrix indicates the average strength and direction found in the literature; +1 indicates the strongest positive correlation and vice versa -1. Values of zero indicates both no effect or that the effect was not assessed in the literature (*notice* that, with respect to the supplementary material in Hanisch et al. (2020), for simplicity we added *zeros* to all empty cells for the analyses shown in this exercise).

We then considered part of the data from the experiment described by Galland et al. (2019). The data used here, provided in the R object 'Benesov.r', include different objects, within a *list*. Most important it includes biomass value of *19 plant species sown* in monocultures and mixtures in a biodiversity experiment. Each community (both monocultures and mixtures) was sown, after plowing, in plots of 1.5mx1.5 m, at at location called *Benesov*, in the central part of Bohemia, in Czech Republic.

In the mixtures (40 plots), different combinations of 6 species, out the original 19 species, were selected, with the aim to have communities with uncorrelated values of functional diversity, phylogenetic diversity and community weighted means (CWMs) for different traits, following the design of experiments described in Dias et al. (2013) and further described in the reference textbook. The data imported below, is an R list with different type of information and more precisely: `$mono`=the biomass of the plant species in the monocultures (average of 3 monocultures); `$mix`=the biomass observed in the mixtures, for each species and each of the mixtures; `$sown`=the proportion of each species in the mixture at the time of sowing (since we sown 6 species, with similar abundance, the values are 1/6=0.167 for each species in the plot where the species was sown); `$predictors`=these are the values of functional diversity, phylogenetic diversity and community weighted mean, which are further detailed below; `$EFs`=a number of parameters associated, directly or indirectly, to different ecosystem functions, see more details below.

```
load(here::here("data", "chapter9","Benesov.r"))
class(Benesov)
```

```
## [1] "list"
```

```
names(Benesov)
```

```
## [1] "mono"       "mix"        "sown"       "predictors" "EFs"
```

9.2 Required packages and additional functions

We need now to upload the package `ade4` for running a PCA, and we can use `factoextra` to visualize nicely the results. Another package for multivariate analyses is `vegan`, which we will use for RDA analyses (although it can be also used for PCA). We also load an ad-hoc function, created by Thomas Galland, to apply the decomposition of *net diversity effects* into *selection* and *complementarity effects*, following the method originally proposed by Loreau & Hector (2001). See the reference textbook, and below, for further details. Finally, we play with some simulations, and indices of CWM and FD, for which we need the `FD` package and the package `tidyverse` as well to run the function for decomposing the net diversity effect.

```
library(ade4)
library(vegan)
library(factoextra)
library(FD)
library(tidyverse)
source(here::here("data", "chapter9", "partBio_Rfct.r"))
```

9.3 Multivariate analyses between traits and ecosystem functions

A simple, yet powerful, way to visualize the multiple connections between traits and/or ecosystem functions is through *multivariate analyses*. The aim of this type analyses is to visualize how different traits associate to multiple ecosystem functions, in order to identify groups of traits associated to groups of ecosystem functions. By this we can observe possible *trade-offs and synergies* between traits and ecosystem functions. Be aware that multivariate analyses are based on linear correlations and therefore not necessarily showing cause-effect relationships. Indeed multivariate analyses are not the only possible way to explore this and, for example, Hanisch et al. (2020) used network analyses for this purpose. However we think multivariate analyses are a useful tool to explore the main existing relationships, at least as a first descriptive and visual step.

Let's first look at the data from Hanisch et al. (2020), at least for the first 3 columns:

```
head(multiEFs[, 1:3])
```

```
##                                Biomass.Production Soil.Fertility
## Leaf nitrogen content (LNC)                  0.50           0.42
## Plant height (VPH)                           0.50           0.66
## Specific leaf area (SLA)                     0.57          -0.06
## Leaf dry matter content (LDMC)              -0.50           0.17
## Plant C/N                                    0.00          -1.00
## Root depth                                   0.00           1.00
##                                Erosion.Control
## Leaf nitrogen content (LNC)              -0.50
## Plant height (VPH)                        0.00
## Specific leaf area (SLA)                 -0.50
## Leaf dry matter content (LDMC)            0.33
## Plant C/N                                 0.00
## Root depth                               -1.00
```

The data include the strength of the relationship between several traits (rows) and several ecosystem functions (columns). This data can be visualized, for example, using a PCA, showing the relationships between traits and ecosystem functions. Remind that, for simplicity, for this exercise we replaced all *empty cells* in the data from Hanisch et al. (2020) (indicating that the relationship is unknown so far) with zeros. Instead of adding missing values (NA), for non-available data, we decided to use zero (see below for some other ideas on how analyse the data generally providing the same result). Both if we keep the missing cells as NA, or we replaced them with zeros it is clear that some traits were assessed much more than others and that some ecosystem functions also were much more studied than others. This means that some traits and some ecosystem functions had too few cases/observations so that it is safer to exclude them from the analyses, in this case.

```
rowSums(multiEFs != 0) #number of entries in the table, different from zero, for each trait
```

```
##       Leaf nitrogen content (LNC)                Plant height (VPH)
##                                 9                                 7
##          Specific leaf area (SLA)    Leaf dry matter content (LDMC)
##                                 8                                 6
##                          Plant C/N                         Root depth
##                                 4                                 4
##       Leaf phosphat content (LPC)                Flowering duration
##                                 4                                 4
##                         Leaf area                  Leaf Litter Mass
##                                 4                                 3
##         Plant N content (Plant N)             Leaf tensile strength
##                                 3                                 3
##         Leaf carbon content (LCC)                       Root length
##                                 3                                 3
##        Specific root length (SRL)                         Lignin/N
##                                 3                                 3
##                     Root diameter                    Leaf toughness
##                                 3                                 2
##               Root tissue density                       Root/shoot
##                                 2                                 2
## Stem dry matter content (StemDMC)             Stem specific density
##                                 2                                 2
##                  Canopy structure            Beginning of flowering
##                                 2                                 2
##          Nitrogen fixing capacity        Relative growth rate (RGR)
##                                 2                                 2
##                    Plant dry mass                          Leaf C/N
##                                 2                                 2
##                      Flower color                       Seed number
##                                 2                                 2
##                       Growth form            Root N content (RootN)
##                                 3                                 1
##    Root dry matter content (RDMC)           Plant height generative
##                                 1                                 1
##        Flower pollination syndrom        Root distribution evenness
##                                 1                                 1
##                      Root density             Percentage fine roots
##                                 1                                 1
##               Root Length Density                         Seed mass
##                                 1                                 0
```

```
colSums(multiEFs != 0) #number of entries in the table, different from zero, for each function
```

```
##  Biomass.Production      Soil.Fertility     Erosion.Control  Climate.Regulation
##                  22                  18                  15                  18
##      Fodder.Quality    Aesthetic.Appeal    Water.Regulation         Pollination
##                   9                  10                   7                   4
## Recreation.Heritage          Biocontrol  Water.Purification
##                   4                   2                   2
```

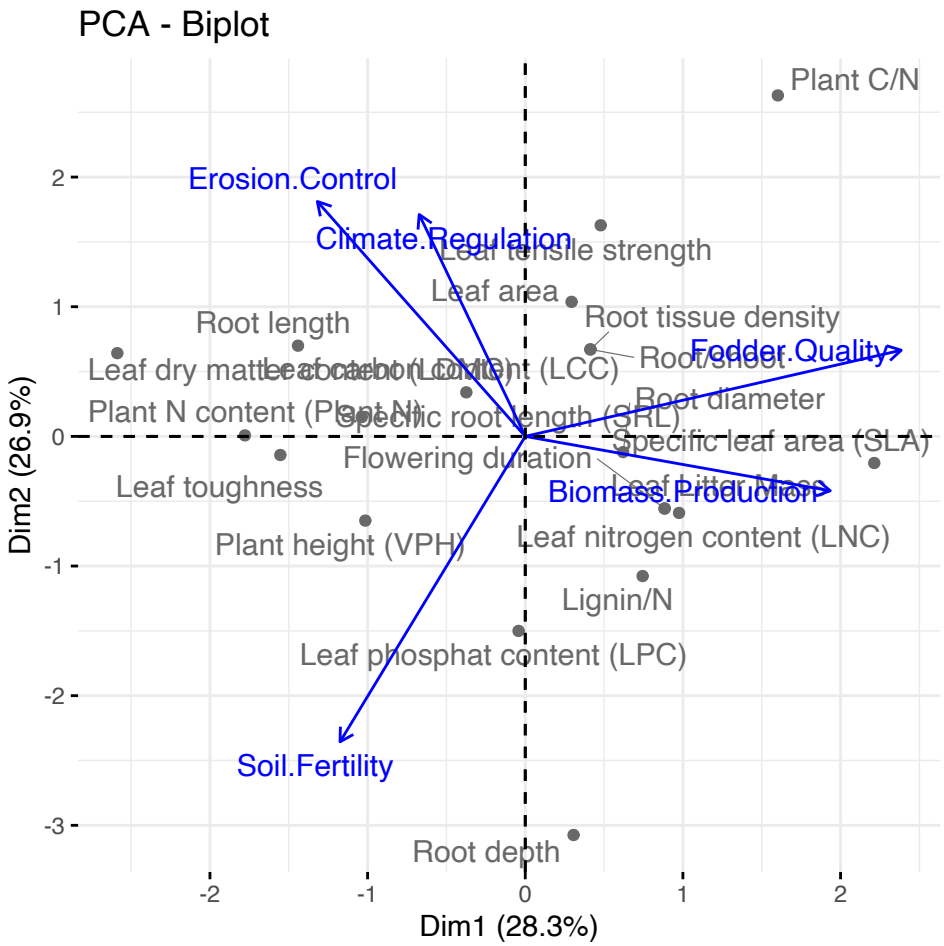
For illustration purposes, we focus here only on the 20 traits and the five ecosystem functions with the most entries in the data set. Of course, this analysis is only meant as an example of the potential analyses that could be done with more robust data.

```
mutlispca <- dudi.pca(multiEFs[1:20, c(1:5)], scannf = FALSE, nf = 2)
```

Notice that here, we use `scannf = FALSE` and `nf = 2` because for simplicity we want only to keep the two most important axes, and we do not want to be asked how many axes we

want to retain. We can then visualize the PCA as follows (thus producing the same figure as Fig. 9.3 in the reference textbook):

```
fviz_pca_biplot(mutlispca, repel = TRUE,
                col.var = "blue", # Variables color
                col.ind = "#696969") # Individuals color
```



The corresponding PCA shows that the first two axes account for 28% and 27% (Dim1 and Dim2 in the biplot) of total variability in the data (see numbers on next to 'Dim1' and 'Dim2'), so they altogether account for almost 56% of the data variability. The angle between arrows (ecosystem functions) illustrates the *correlation* between the ecosystem functions along these two main axes. So, for example, fodder quality and biomass production are likely positively correlated between them in grasslands, based on the traits associated between them (see also the correlation values between ecosystem functions, below). Also, the position of traits in the biplot indicates their association with ecosystem functions. For example SLA and Leaf Nitrogen Content seem to be well associated with fodder quality and biomass production. Let's now check the correlation between ecosystem functions:

```
round(cor(multiEFs[, c(1:5)]), 2)
```

```
##                    Biomass.Production Soil.Fertility Erosion.Control
## Biomass.Production               1.00           0.03           -0.04
## Soil.Fertility                   0.03           1.00           -0.16
## Erosion.Control                 -0.04          -0.16            1.00
## Climate.Regulation              -0.09           0.02            0.00
## Fodder.Quality                   0.13          -0.14           -0.13
##                    Climate.Regulation Fodder.Quality
```

```
## Biomass.Production              -0.09           0.13
## Soil.Fertility                   0.02          -0.14
## Erosion.Control                  0.00          -0.13
## Climate.Regulation               1.00          -0.02
## Fodder.Quality                  -0.02           1.00
```

Indeed, in this specific case, the correlations between ecosystem functions are generally low, but the correlation between fodder quality and biomass production shows one of the highest correlation values in the correlations found. The correlation is generally low possibly because we are not studying, in this specific case, directly the values in ecosystem functions but how these ecosystem functions are associated to traits. Indeed this is not the best type of data we can use, but it still provides some interesting information for this exercise. See below for other type of more robust data.

As mentioned above, we decided to replace the empty cells in the Hanisch et al. (2020) data with zeros to use PCA. Another option would have been to use NA in the empty cells. Then we could have used Gower distance and run a *PCoA* (see also R material Ch3). We did this and generally obtained similar results, but which were more difficult to visualize. So, for simplicity, we do not shown them here.

The ideal type of data we need for testing the effects of traits on multiple ecosystem functions is another one. Ideally, one could have data for different plots, preferably in a rather uniform environment, each with different measures of ecosystem functions and species traits.

This is exactly the data that is available for the next example that we are going to consider. The data 'Benesov', which we have already loaded above as a `list`, contains several types of data. One, called EFs, contains several parameters which can be considered as ecosystem functions, either directly or potentially associated to several soil ecosystem processes These are data from sown communities with different levels of functional diversity, phylogenetic diversity and CWM, within the same field. Let's first look at the variables/parameters associated with ecosystem functions.

```
names(Benesov$EFs)
```

```
##  [1] "tot_biom"      "tot_biom_weed" "Decomp"        "TOC"
##  [5] "TN"            "P"             "BR"            "gluco"
##  [9] "phospha"       "ure"
```

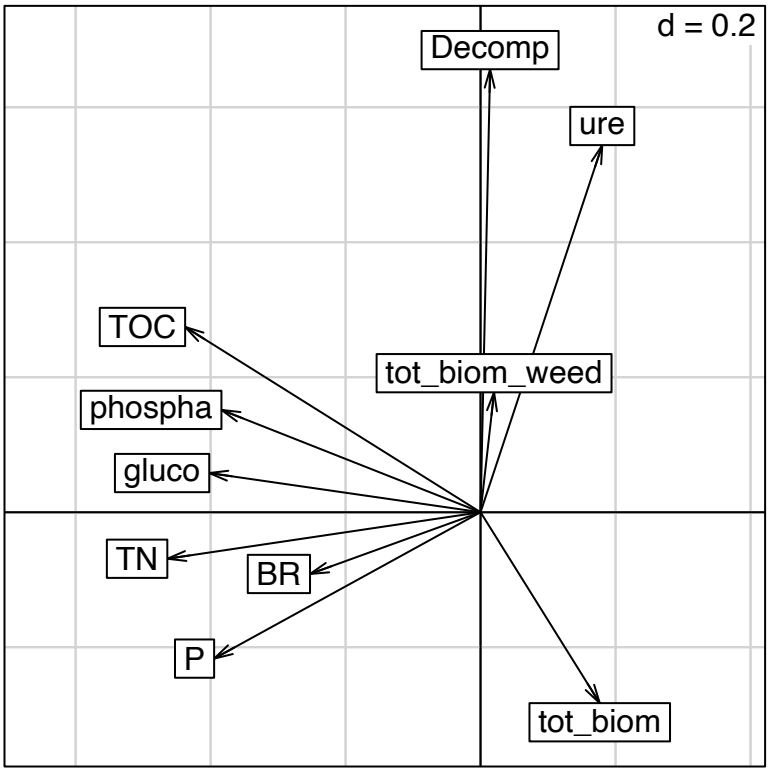
We can now run a simple PCA, exactly as we did before, but this time with real field data and not meta-data. In this case we start with a *correlation* between ecosystem functions related parameters, and then show a *PCA*, which should reflect exactly the structure of the correlation. Notice that this time we use directly the function `s.arrow` because it a simple function which can allow visualizing directly the relationship between the studied parameters (we could clearly could have used the function `fviz_pca_biplot`, see above).

```
as.dist(round(cor(Benesov$EFs), 3))
```

```
##               tot_biom tot_biom_weed Decomp    TOC     TN      P     BR  gluco
## tot_biom_weed   -0.078
## Decomp          -0.161         0.178
## TOC             -0.345        -0.153  0.207
## TN              -0.300         0.027 -0.040  0.691
## P               -0.078         0.014 -0.118  0.509  0.679
## BR              -0.040        -0.240 -0.067  0.332  0.336  0.248
## gluco           -0.115        -0.015 -0.027  0.600  0.498  0.395  0.307
## phospha         -0.147         0.107  0.002  0.536  0.511  0.404  0.239  0.625
## ure              0.097        -0.066  0.329  0.019 -0.401 -0.397 -0.076 -0.171
##               phospha
```

```
## tot_biom_weed
## Decomp
## TOC
## TN
## P
## BR
## gluco
## phospha
## ure             -0.038
```

```
Benesov.pca <- dudi.pca(Benesov$EFs, scannf = FALSE, nf = 2)
s.arrow(Benesov.pca$c1)
```



In the object `Benesov.pca$li`, we will find the scores of the 40 plots considered, along the two main PCA axes. Now we can test the correlation of single parameters present into `Benesov$EFs`, with the two main axes of the PCA. For example:

```
t(cor(Benesov.pca$li, Benesov$EFs))
```

```
##                    Axis1       Axis2
## tot_biom       0.33209520 -0.34960311
## tot_biom_weed  0.03764468  0.21957851
## Decomp         0.02654016  0.81123492
## TOC           -0.82316700  0.33927332
## TN            -0.87278176 -0.08520708
## P             -0.74236944 -0.26807576
## BR            -0.47517905 -0.11307321
## gluco         -0.75596172  0.07174264
## phospha       -0.72200793  0.18738209
## ure            0.33913467  0.67230352
```

By looking at these correlations and the PCA figure done above with `s.arrow`, we see that the first axis is correlated mostly with beta-glucosidase activity (`gluco`, soil enzymatic activity involved in the C cycle), phosphatases (`phospa`, soil enyzmatic activity involved in the P cycle) on the left side of the first axis. This is generally associated with an increase of total soil carbon (`TOC`), total soil nitrogen (`TN`), and soil phosphorous (P). On the other side of the first axis we can see that total plot biomass (`tot_biom`) is increasing, so we can imagine that there is some trade-off, albeit not very strong, between increase in productivity and soil activity related to soil C, N and P. On the second axis we can see that litter decomposability (`Decomp`) and urease activity (`ure`, soil enyzmatic activity involved in the N cycle) are increasing in the same direction and they increase in the opposite direction of greater total biomass. The general correlations between these parameters, which are shown in `cor(Benesov$EFs)`, were not very strong, but at least these type of analyses, including PCAs and correlations, give us a first glimpse of how different soil parameters and ecosystem functions covary between them in our study system.

9.4 Testing the relationship between traits and ecosystem functions

Once we have visualized the main relationships between ecosystem functions, in the previous section, we can try to perform a number of formal tests to relate these functions with trait data, in order to explore the strongest relationships. Of course, the number of possible tests is virtually unlimited, depending on the questions being asked, the experimental design and traits considered. Also, the number of statistical tools that can be used is large (ranging from simple models, multivariate analyses, structural equation models etc.). Here we want to provide some clear examples, based on well-established literature, to relate traits to ecosystem functions.

In this context we will broadly follow the approach proposed by Diaz et al. (2007) in which the authors suggest to consider two components of the functional structure of communities, i.e. functional diversity and community weighted mean (see Chapter 5 in the reference textbook and corresponding R material), i.e. *FD and CWM*. We can also consider *phylogenetic diversity* (PD) as a proxy of unmeasured trait diversity (see Chapter 8 and 9 in the reference textbook and corresponding R material). These predictors can reflect different mechanisms by which the traits in a community can affect ecosystem functions (see Chapter 9 for more theoretical explanations).

In our example here we can see the type of predictors which depend on traits and phylogeny in the `Benesov` data:

```
head(Benesov$predictors)
```

```
##        PDbiom       FDbiom cwm_biom_Dim.1 cwm_biom_Dim.2
## 10E 243.85552 0.0004714775      0.1531378     0.32978541
## 10J 244.19667 0.0007043456      0.1998662    -0.35253754
## 11G 233.02818 0.0005423321      0.1021063     0.16851074
## 12E 230.92060 0.0006361238      0.4558700     0.04907175
## 12I  32.40313 0.0004563644      0.3539096     0.24032821
## 13A 206.79599 0.0006052323      0.2646602     0.42895831
```

The 40 plant mixtures considered in this dataset were assembled, by sowing communities with high or low phylogenetic diversity, in a quasi-orthogonal design, so that all possible combinations of high and low PD and FD could be found. Two years after the sowing, biomass was collected in each plot and sorted into species. Using the observed biomass per species we computed different indices, using species relative abundance expressed as proportion of the total biomass (see Chapter 5 and corresponding R material). PDbiom reflects the phylogenetic diversity observed in the plot at the time of sampling, using the Rao index.

Similarly FDbiom was computed also using the Rao index, with several traits measured at each site (plant height, and four leaf traits, i.e. SLA, LDMC, P and C:N; notice: the correlation between these traits was below R=0.6, Pearson correlation) and considering trait overlap (Rao index?!) as a measure of dissimilarity (see Chapter 3 and 6, and corresponding R materials, for details). The two variables `cwm_biom_Dim.1` and `cwm_biom_Dim.2` are the community weighted means considered for this example. In practice, since several traits were measured for the species sown, we first run a *PCA on the species x trait matrix* (see Chapter 3 and 5, and corresponding R material) and then retained the species scores on the first two axes (Dim1 and Dim2). The species scores on these two axes were used to compute CWMs. The positive values in the first axis broadly reflect taller species with greater LDMC and positive values in the second axis reflects high leaf C:N ratio and low SLA.

The first thing we can do, if we are interested in a specific ecosystem function (in `Benesov$EFs`) is to see which predictors (in `Benesov$predictors`) are best related to it. For example, in case of *productivity*, i.e. total biomass, we can use a step-wise linear regression,

```
summary(step(lm(Benesov$EFs$tot_biom ~ ., data = Benesov$predictors)))
```

```
## Start:  AIC=-101.42
## Benesov$EFs$tot_biom ~ PDbiom + FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq    RSS      AIC
## - PDbiom          1   0.08474 2.5529 -102.065
## <none>                        2.4682 -101.416
## - FDbiom          1   0.56802 3.0362  -95.131
## - cwm_biom_Dim.2  1   0.63087 3.0991  -94.311
## - cwm_biom_Dim.1  1   1.02378 3.4920  -89.536
##
## Step:  AIC=-102.07
## Benesov$EFs$tot_biom ~ FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq    RSS      AIC
## <none>                        2.5529 -102.065
## - cwm_biom_Dim.2  1   0.67209 3.2250  -94.718
## - FDbiom          1   0.71687 3.2698  -94.166
## - cwm_biom_Dim.1  1   1.16782 3.7208  -88.998

##
## Call:
## lm(formula = Benesov$EFs$tot_biom ~ FDbiom + cwm_biom_Dim.1 +
##     cwm_biom_Dim.2, data = Benesov$predictors)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.49831 -0.15419  0.01313  0.19309  0.50079
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3.5472     0.4099   8.654 2.55e-10 ***
## FDbiom         2031.7315   639.0202   3.179 0.003030 **
## cwm_biom_Dim.1    0.5730     0.1412   4.058 0.000254 ***
## cwm_biom_Dim.2    0.4041     0.1313   3.079 0.003966 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2663 on 36 degrees of freedom
## Multiple R-squared:  0.4552, Adjusted R-squared:  0.4098
## F-statistic: 10.03 on 3 and 36 DF,  p-value: 6.077e-05
```

The results suggest that productivity (`tot_biom`) is significantly related to the both CWM parameters. This means that productivity increases when the dominant species are taller (obviously), and with higher LDMC and higher C:N ratio in the leaves. However productivity also increases with greater functional diversity within plots. The model explains around 40% (adjusted R-squared) of the total variability, which is quite a decent model.

We can repeat the same thing, but considering all EFs parameters in `Benesov$EFs` at once, using multivariate approaches like RDA. If you are not very familiar with this approach, please consider this very interesting and relative simple page by David Zeleny, with corresponding R tools (https://www.davidzeleny.net/anadat-r/doku.php/en:rda_cca). In the RDA, instead of running a single linear model for each EFs parameter we do one single model where *ALL* the EFs are considered in response to the predictors:

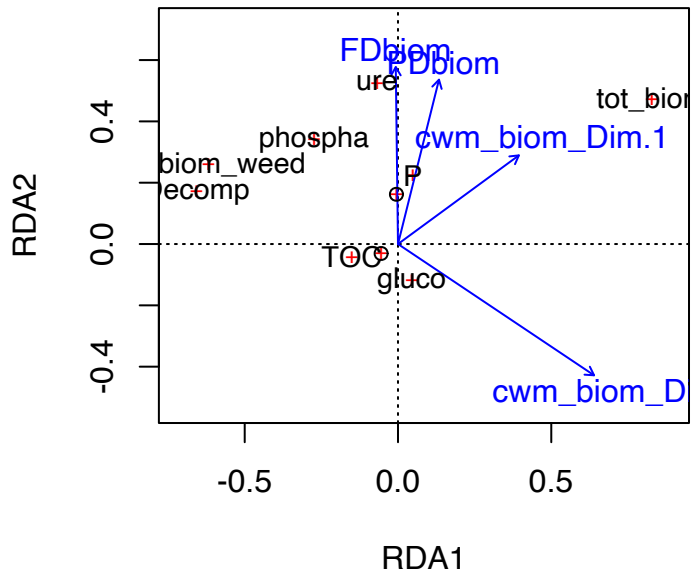
```
rdaBenesov <- rda(scale(Benesov$EFs) ~ ., data = Benesov$predictors)
anova(rdaBenesov)
```

```
## Permutation test for rda under reduced model
## Permutation: free
## Number of permutations: 999
##
## Model: rda(formula = scale(Benesov$EFs) ~ PDbiom + FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2, da
##          Df Variance      F Pr(>F)
## Model     4   1.4783 1.5179  0.069 .
## Residual 35   8.5217
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
R2bene <- RsquareAdj(rdaBenesov)$adj.r.squared
R2bene
```

```
## [1] 0.05043928
```

```
ordiplot(rdaBenesov, display = c('sp', 'cn'))
orditorp(rdaBenesov, display = 'sp', cex = 0.9)
```



In this analysis we can see that the overall RDA is only marginally significant (using `anova(rdaBenesov)`. Notice that the p-value can change because the test is run using permutations). Similarly the model fit (sing `RsquareAdj`) was generally low, ~0.05, i.e. the

model explains 5% (keep in mind that the linear model only using `Benesov$EFs$tot_biom` was around 40%, see above). So, in principle this tells us that the EFs considered are pretty unrelated between them, without any strong trade-offs associated to the predictors considered. If we focus on the plot displayed above we can see that there is anyway some suggestion that the first RDA axis is associated with a *trade-off* between productivity (`tot_biom`) and invasion from weed species and decomposition, associated with an increased in the CWM values considered, see above. The second axis is also partially associated with productivity and N cycle (urease), associated with greater FD and PD.

We can also test, as we did above using `lm`, if all predictors are actually retained in a *parsimonious* RDA model. For this using a similar concept of *variable selection* in `lm`. This is done by comparing an 'empty' model, without predictors, with one with all predictors, and the function `ordistep`, for example (other functions also work fine):

```
rdaBenesov.null <- rda(scale(Benesov$EFs) ~ 1, data = Benesov$predictors)
rdaBenesov <- rda(scale(Benesov$EFs) ~ ., data = Benesov$predictors)
sel <- ordiR2step (rdaBenesov.null, scope = formula (rdaBenesov),
                   R2scope = R2bene, direction = 'forward', permutations = 999)
```

```
## Step: R2.adj= 0
## Call: scale(Benesov$EFs) ~ 1
##
##                     R2.adjusted
## <All variables>   0.0504392795
## + cwm_biom_Dim.2  0.0393861086
## + cwm_biom_Dim.1  0.0084427432
## + FDbiom          0.0044297411
## <none>            0.0000000000
## + PDbiom         -0.0007994084
##
##                   Df    AIC     F Pr(>F)
## + cwm_biom_Dim.2   1 92.444 2.599  0.023 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step: R2.adj= 0.03938611
## Call: scale(Benesov$EFs) ~ cwm_biom_Dim.2
##
##                   R2.adjusted
## + FDbiom           0.05133123
## <All variables>    0.05043928
## + cwm_biom_Dim.1   0.04486141
## + PDbiom           0.03957733
## <none>             0.03938611
```

```
sel$anova
```

```
##                     R2.adj Df    AIC     F Pr(>F)
## + cwm_biom_Dim.2 0.039386  1 92.444 2.599  0.023 *
## <All variables>  0.050439
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This forward selection, selected only one predictor, i.e. CWM based on the second PCA of species traits (with higher values indicating lower SLA and higher leaf C:N). This seems to suggest that this is the only real trait component which create some sort of trade-offs between the ecosystem functions considered. This could be visualized like this:

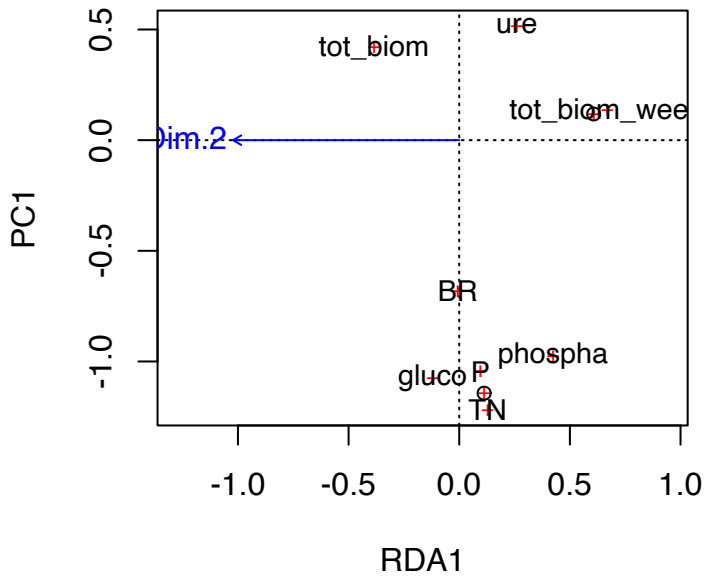
```
rdaBenesov.CWMaxis2 <- rda(scale(Benesov$EFs) ~ Benesov$predictors$cwm_biom_Dim.2)
anova(rdaBenesov.CWMaxis2)
```

```
## Permutation test for rda under reduced model
## Permutation: free
## Number of permutations: 999
##
## Model: rda(formula = scale(Benesov$EFs) ~ Benesov$predictors$cwm_biom_Dim.2)
##          Df Variance     F Pr(>F)
## Model     1   0.6402 2.599  0.013 *
## Residual 38   9.3598
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
R2bene.CWMaxis2 <- RsquareAdj (rdaBenesov.CWMaxis2)$adj.r.squared
R2bene.CWMaxis2
```

```
## [1] 0.03938611
```

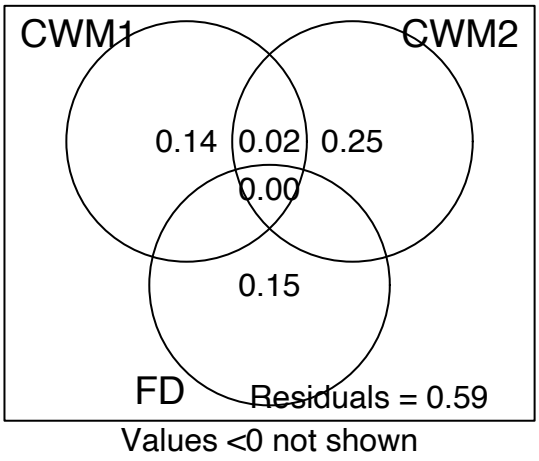
```
ordiplot(rdaBenesov.CWMaxis2, display = c('sp', 'cn'))
orditorp(rdaBenesov.CWMaxis2, display = 'sp', cex = 0.9)
```



This shows a significant, albeit weak, model (3% explained of overall variability) in which `cwm_biom_Dim.2` is related positively to productivity and negatively to urease and weed biomass.

Similar approaches can be applied to different datasets to test the relative effect of different variables. For example, if we consider only productivity, we can test the relative effect of CWM and FD components. *Notice* that this is still done with an RDA, which in the case of a single dependent variable, like here, basically becomes a linear model similar to the one we used above. The advantage here is that, while the `lm` provides only marginal effects (when multiple predictors are considered), the `varpart` provide both marginal and absolute effect, thus allowing visualizing the shared effect of different variables, with a *Venn diagram*:

```
varp <- varpart(Benesov$EFs$tot_biom, ~ Benesov$predictors$cwm_biom_Dim.2,
                ~ Benesov$predictors$cwm_biom_Dim.1, ~ Benesov$predictors$FDbiom)
plot(varp, Xnames = c('CWM1', 'CWM2', 'FD'))
```



In this case we decided to not use this when considering all EFs, i.e. `Benesov$EFs`, at once. This is because we already shown that only one predictor was significant, and weakly, when considered the ensemble of all EFs in this dataset.

9.5 Decomposing the net diversity effect

In the previous section we have studied, among other things, how the overall productivity of the sown communities changed with species traits. We have seen that productivity was related to both CWM and FD components and this already allowed us to draw some key biological conclusions, following the framework proposed by Diaz et al. (2007); see Chapter 9 in the reference text book for more details].

Following the literature from BEFs experiments we also know that biodiversity effects on ecosystem functions can be decomposed into two mechanisms, *selection effect* (the effect a given species, whose presence and abundance drive biodiversity effects) and *niche complementarity* (the positive non-additive effect of coexistence, i.e. when the effect provided by some species in the mixtures is higher that the effects provided by the sum of the single performances in the monocultures). In the mathematical framework proposed by Loreau & Hector (2001), *selection effect + niche complementarity = net diversity effect*. Positive net diversity effects, or overyielding, implies that in average species in mixture overall grow more (or in more general terms 'perform' better) more that when they are grown alone, in monocultures. using a specific equation, which we do not explain here for simplicity, Loreau & Hector (2001), one can thus decompose net diversity effect into selection effect and niche complementary components. To apply this formula we need species performance, in our case (and often elsewhere) "biomass", both in mixtures and monocultures. The object `Benesov$mono` includes the average biomass of the 19 species sown in the Benesov experiment, over 3 replicated plots.

```
Benesov$mono #notice that "abund" is plant biomass
```

```
##             abund
## Ach_mil 236.60400
## Alo_pra  73.72667
## Ant_odo  85.04000
## Ant_vul  17.58187
## Dac_glo  64.46933
## Dia_del  67.54433
## Hol_lan  64.57833
## Hyp_per 131.18267
## Leo_his  67.46100
## Leu_vul  77.47900
```

```
## Lot_cor 102.64450
## Lyc_flo  25.21550
## Pla_lan  78.40433
## Pla_med  29.76567
## Poa_pra  24.50733
## Pru_vul  61.51033
## Tri_arv   8.31210
## Tri_pra  25.23567
## Vic_sep  16.93033
```

We can then look at the biomass observed in the 40 sown communities, at the second year after sowing, for these 19 species:

```
head(Benesov$mix)
```

```
##     Ach_mil Alo_pra Ant_odo Ant_vul Dac_glo Dia_del Hol_lan Hyp_per Leo_his
## 10E       0  1.9877  46.294  0.0000   0.000  24.917   0.000       0  2.7105
## 10J       0  0.0000   0.000  0.0000  14.864  42.578  49.522       0  0.0000
## 11G       0  0.0000  24.439  2.1653   0.000   0.000  30.921       0  0.0000
## 12E       0  0.0000   0.000  0.0000   0.000   5.545   0.000       0  0.4458
## 12I       0  1.6990  33.020  0.0000  16.033   0.000  16.956       0  0.0000
## 13A       0  0.0000  17.490  3.9960  12.165   0.000  21.837       0  0.0000
##     Leu_vul Lot_cor Lyc_flo Pla_lan Pla_med Poa_pra Pru_vul Tri_arv Tri_pra
## 10E 27.7790   0.000  0.0000   0.000  0.0000       0       0       0       0
## 10J  0.0000  39.268  0.0000   0.000  3.5640       0       0       0       0
## 11G 56.8850   0.000  0.0097   0.000  0.0761       0       0       0       0
## 12E  0.3665  90.823  0.0000  52.572  0.0000       0       0       0       0
## 12I  0.0000   0.000  0.0000   0.000  0.0000       0       0       0       0
## 13A  0.0000   0.000  0.0000  43.255  0.0000       0       0       0       0
##     Vic_sep
## 10E  0.0000
## 10J  6.8983
## 11G  0.0000
## 12E  0.0000
## 12I  0.0000
## 13A  0.0000
```

We can then apply the function made by Thomas Galland, our former PhD student, to apply the Loreau & Hector (2001) formula with this type of data:

```
div_effect <- partBio(monoYield = Benesov$mono,
                      obsYield = Benesov$mix,
                      propSown =  Benesov$sown)
```

Notice that we had also to specify the sown proportion of the different species (in `Benesov$sown`. In our case, we attempted to have a similar starting biomass for the different species (see Galland et al., 2019), and so we provide this as a equal sown proportion for each of the 6 species sown in each mixture (1/6=0.167).

We can now look at the object produced above:

```
head(div_effect)
```

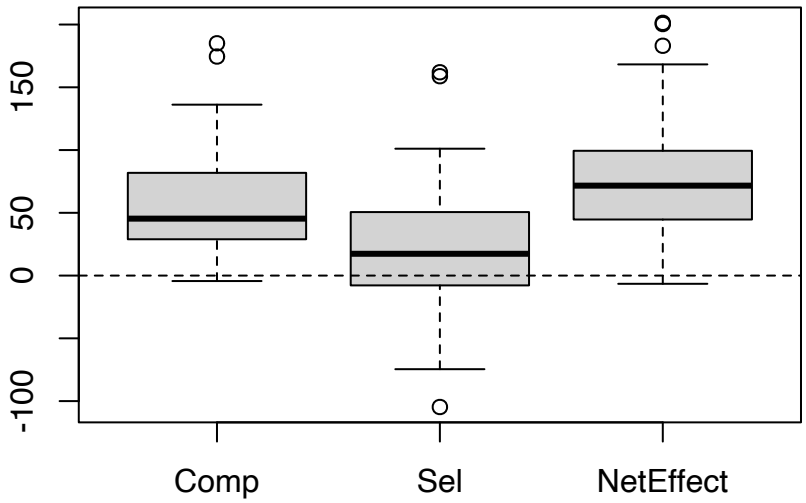
```
##          Comp       Sel NetEffect     Trans     maxM
## 10E 28.383532 -8.434277  19.94926 0.7904108 131.1827
## 10J 88.647078 10.391806  99.03888 1.5265728 102.6445
## 11G 31.289128 33.263578  64.55271 1.3463794  85.0400
```

```
## 12E 56.738190  5.561471  62.29966 1.1415555 131.1827
## 12I -4.354839 15.806644  11.45181 0.7961900  85.0400
## 13A 28.970234 13.132844  42.10308 1.1611359  85.0400
```

This output includes niche complementarity effect (`Comp`), selection effect (`Sel`) and Net diversity effect (`NetEffect`). It also includes *transgressive overyielding* (`Trans`), which is the relationship between the observed biomass with respect to the one expected from the most productive monoculture of the species composing the mixture. Finally, it includes `maxM` which indicates the maximum biomass in the monoculture among the species composing the mixture.

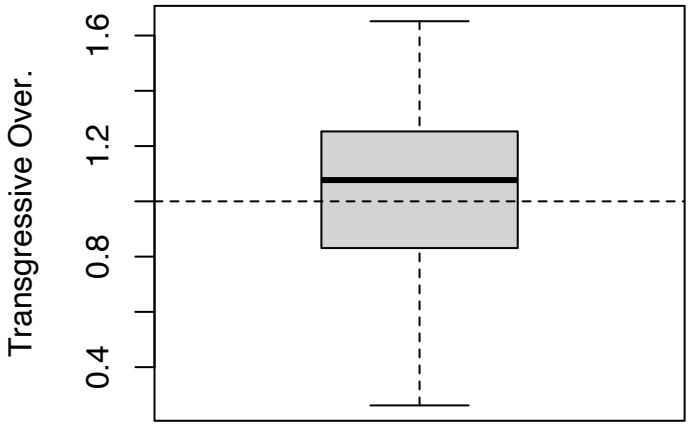
In our analyses we can see different things. In general net diversity effects are positive, so that mixtures perform better that monocultures, and that is due to both positive effect of complementarity effects (more strongly) and selection effects (less strongly). This can be, for example, visualized as:

```
boxplot(div_effect[, 1:3])
abline(h = 0, lty = 2)
```



Also in some cases the mixture performs even more than the most productive monoculture (transgressivity). Values higher than 1 would indicate a strong transgressive overyielding. In our case transgressive overyielding, values are not sufficiently, and consistently, greater than 1 (see next figure). This means that carefully chosen monocultures can be often as productive as mixtures.

```
boxplot(div_effect[, 4], ylab = "Transgressive Over.")
abline(h = 1, lty = 2)
```



```r
t.test(div_effect[, 4] - 1) #test on whether "Transgressive Over." is different from 1.
```

```
##
##  One Sample t-test
##
## data:  div_effect[, 4] - 1
## t = 1.0217, df = 39, p-value = 0.3132
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.04946563  0.15044095
## sample estimates:
##  mean of x
## 0.05048766
```

Then, we can relate the net diversity, complementarity and selection effects to traits, similarly as the way we did when we tested the effect of traits and phylogeny on total biomass, above:

```r
summary(step(lm(div_effect[, 1] ~ ., data = Benesov$predictors))) #complementarity
```

```
## Start:  AIC=303.29
## div_effect[, 1] ~ PDbiom + FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                   Df Sum of Sq   RSS    AIC
## - cwm_biom_Dim.2   1     487.8 61634 301.60
## - PDbiom           1     871.8 62018 301.85
## - cwm_biom_Dim.1   1    1956.6 63103 302.55
## <none>                         61147 303.29
## - FDbiom           1    6656.3 67803 305.42
##
## Step:  AIC=301.6
## div_effect[, 1] ~ PDbiom + FDbiom + cwm_biom_Dim.1
##
##                   Df Sum of Sq   RSS    AIC
## - PDbiom           1     776.7 62411 300.11
## - cwm_biom_Dim.1   1    1863.6 63498 300.80
## <none>                         61634 301.60
## - FDbiom           1   10891.5 72526 306.11
##
## Step:  AIC=300.1
## div_effect[, 1] ~ FDbiom + cwm_biom_Dim.1
##
##                   Df Sum of Sq   RSS    AIC
## - cwm_biom_Dim.1   1    2408.2 64819 299.62
## <none>                         62411 300.11
## - FDbiom           1   12918.0 75329 305.63
##
## Step:  AIC=299.62
## div_effect[, 1] ~ FDbiom
##
##           Df Sum of Sq   RSS    AIC
## <none>                 64819 299.62
## - FDbiom   1     11924 76744 304.37

##
## Call:
```

```
## lm(formula = div_effect[, 1] ~ FDbiom, data = Benesov$predictors)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -58.62 -29.69  -7.26  18.43 122.22
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -84.46      54.41  -1.552   0.1289
## FDbiom      230351.12   87123.14   2.644   0.0118 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 41.3 on 38 degrees of freedom
## Multiple R-squared:  0.1554, Adjusted R-squared:  0.1332
## F-statistic: 6.991 on 1 and 38 DF,  p-value: 0.01184
```

```
summary(step(lm(div_effect[, 2] ~ ., data = Benesov$predictors))) #selection effect
```

```
## Start:  AIC=313.39
## div_effect[, 2] ~ PDbiom + FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq    RSS    AIC
## - PDbiom          1      27.5  78748 311.40
## - FDbiom          1     465.1  79185 311.63
## <none>                         78720 313.39
## - cwm_biom_Dim.1  1    7988.6  86709 315.26
## - cwm_biom_Dim.2  1   21666.7 100387 321.12
##
## Step:  AIC=311.4
## div_effect[, 2] ~ FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq    RSS    AIC
## - FDbiom          1     437.6  79185 309.63
## <none>                         78748 311.40
## - cwm_biom_Dim.1  1    8077.4  86825 313.31
## - cwm_biom_Dim.2  1   21680.0 100428 319.13
##
## Step:  AIC=309.63
## div_effect[, 2] ~ cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq    RSS    AIC
## <none>                         79185 309.63
## - cwm_biom_Dim.1  1    7882.8  87068 311.42
## - cwm_biom_Dim.2  1   24190.8 103376 318.29

##
## Call:
## lm(formula = div_effect[, 2] ~ cwm_biom_Dim.1 + cwm_biom_Dim.2,
##     data = Benesov$predictors)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -91.501 -24.168   3.739  23.410 128.987
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)     -0.8799    11.4657  -0.077  0.93924
## cwm_biom_Dim.1  46.9899    24.4842   1.919  0.06270 .
## cwm_biom_Dim.2  67.7465    20.1504   3.362  0.00181 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 46.26 on 37 degrees of freedom
## Multiple R-squared:  0.309,  Adjusted R-squared:  0.2717
## F-statistic: 8.273 on 2 and 37 DF,  p-value: 0.001072
```

```
summary(step(lm(div_effect[, 3] ~ ., data = Benesov$predictors))) #net diversity effect
```

```
## Start:  AIC=304.62
## div_effect[, 3] ~ PDbiom + FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq   RSS    AIC
## - PDbiom          1     539.2 63756 302.96
## <none>                        63217 304.62
## - FDbiom          1   10771.7 73989 308.91
## - cwm_biom_Dim.2  1   15075.7 78293 311.17
## - cwm_biom_Dim.1  1   16292.9 79510 311.79
##
## Step:  AIC=302.96
## div_effect[, 3] ~ FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2
##
##                  Df Sum of Sq   RSS    AIC
## <none>                        63756 302.96
## - FDbiom          1     12653 76409 308.20
## - cwm_biom_Dim.2  1     15627 79383 309.73
## - cwm_biom_Dim.1  1     17927 81683 310.87
```

```
##
## Call:
## lm(formula = div_effect[, 3] ~ FDbiom + cwm_biom_Dim.1 + cwm_biom_Dim.2,
##     data = Benesov$predictors)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -79.078 -33.836  -2.518  27.078  97.565
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -117.58      64.77  -1.815  0.07782 .
## FDbiom         269920.91  100984.82   2.673  0.01123 *
## cwm_biom_Dim.1     70.99      22.31   3.182  0.00301 **
## cwm_biom_Dim.2     61.61      20.74   2.970  0.00527 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 42.08 on 36 degrees of freedom
## Multiple R-squared:  0.3741, Adjusted R-squared:  0.3219
## F-statistic: 7.171 on 3 and 36 DF,  p-value: 0.0006778
```

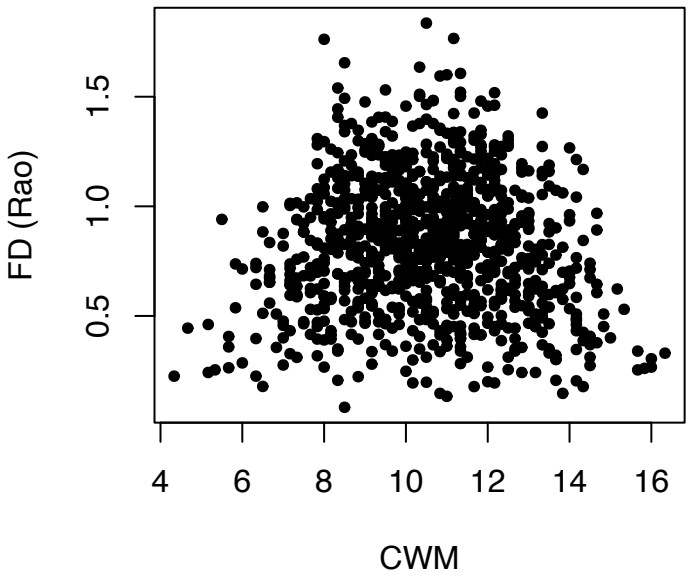
While for the model on net diversity effect, as for total biomass above, both CWM and FD had a shared effect, FD was the only predictor for complementarity and CWM were the only predictor for selection effects. These results follow *exactly* the predictions that complementarity should occur because functionally different species coexist, while selection

effect should occur when some specific species, with given traits, dominate the community, which should be reflect by the predominance of CWM (see also Cadotte, 2017). Interestingly PD dropped from the significant results, most likely because the important traits explaining above ground biomass and the diversity effect on biomass were well accounted for in FD, and/or not well conserved phylogenetically.

9.6 Care with FD and CWM as predictors

Users should be aware that in many natural communities FD and CWM values are not independent between each other (see also Ricotta and Moretti, 2011). We show this with some simple simulations, in which we create 1000 communities from a pool of 20 species. Each community was built by randomly selecting 6 species from the species pool. This means that all the communities had the same species richness. Of course you can play with this example and change the parameters (number of species etc.).

```
Nsp <- 20 # number of species in a pool
Nsim <- 1000 # number of communities we want to simulate
rich <- 6 # richness per plot/community
spxp.test <- matrix(0, Nsim, Nsp) #empty community data
spxt.test <- 1:Nsp #trait values, 1 to the number of species for simplicity.
spxt.matrix <- as.matrix(spxt.test)
row.names(spxt.matrix) <- 1:Nsp
colnames(spxt.matrix) <- "trait"
colnames(spxp.test) <- 1:Nsp
#using a loop to create 1000 communities
for(i in 1:Nsim){
  spi <- sample(spxt.test, rich)
  spxp.test[i, spi] <- 1
}
res <- dbFD(spxt.matrix, spxp.test, messages = F) #computing indices
cwm <- functcomp(spxt.matrix, spxp.test)
plot(res$CWM[, 1], res$RaoQ, pch = 20, xlab = "CWM", ylab = "FD (Rao)")
```



This shows that at highest and lowest levels of CWM it is impossible to have high level of functional diversity, therefore CWM and FD are not independent, and these two related predictors might cause problems when jointly tested in linear models. This means that care should be taken when testing the relative effect of these predictors in natural communities.

In our previous example ('Benesov') we actually made sure that FD and CWM were unrelated, following the paper by Dias et al. (2013). This was done by carefully designing specific mixtures before sowing them. But this is often not the case with observed communities in natural conditions, so that before analyzing FD and CWM authors should allow reviewers to visualize the relationship between these two potential predictors.

10 – Traits and trophic interactions

As discussed in Chapter 10 of the reference textbook, there are several ways to use traits for better understanding species interactions. In this exercise we will present two approaches linking traits to species interactions, one at the species level and another at the community level (see R material 4 and 5 for more examples of trait analyses at species and community levels). In both examples, we will deal with data sets containing traits of grasshoppers and plants. First, at the species level, we will test if (and which) traits of both grasshoppers and plants constrain the grasshopper diet. For this, we will calculate the functional composition of grasshopper food preferences. Second, at the community level, we will use structural equation models (SEM) to test the response-effect trait framework across trophic levels. For this, we will use data from community functional composition of plants and grasshoppers along a gradient of management intensity. By no means these two approaches are exhaustive. As highlighted in Chapter 10 of the reference textbook, the use of species traits for studying species interactions and interaction networks is a fast-growing field of research.

10.1 Data

For investigating the roles of traits in determining grasshopper diet, we will use the data from grasshopper food preference from Ibanez et al. (2013). The data set is composed by a food preference matrix (foodpreference.txt), a plant trait matrix (plantxtraits.txt) and a grasshopper trait matrix (grasshopxtraits.txt). Make sure you have all these files in your working directory.

```
foodpref <- read.table(here::here("data", "chapter10", "foodpreference.txt"), header = T, ro
plantxtraits <- read.table (here::here("data", "chapter10","plantxtraits.txt"), header = T,
grasshopxtraits <- read.table (here::here("data", "chapter10","grasshopxtraits.txt"), header
```

For testing the response-effect trait framework, we will use the data from agricultural fields representing a grandient for management intensity. This data set is based on the data presented by Moretti et al. (2013). Make sure you have this file in your working directory.

```
plantxgrasshop <- read.table (here::here("data", "chapter10","plantxgrasshop.txt"), header =
```

10.2 Required packages

For these exercises we will need two packages. For calculating the functional aspects of the food preference of grasshoppers, we will use the *FD* package. With this package we can easily calculate community functional metrics like *CWM* and *functional diversity indices*. For more details on community functional metrics you can see Chapter 5 of the reference book and its accompanying R material. We will use structural equation models to operationalize the response-effect framework with trophic interactions. This modeling will be performed using the *sem* package. Make sure you have these packages and its dependancies installed.

```
library(FD)
library(sem)
```

10.3 Calculating functional aspects of food preference

For a given species A, the functional component of its diet considers the trait values of all species with which species A trophically interacts. Here we are going to exemplify this concept by calculating the average trait values (CWM) of the plants eaten by different grasshopper species, where the weights are given by the amount of consumed leaf of the different plant species. This matrix of food preference reflects potential plant-grasshopper interactions and was built using a cafeteria experiment, where leaves of several plants were offered to single grasshopper species at a time (Ibanez et al., 2013). With this experiment, it is possible to measure the food preference of each grasshopper species. Adding traits of both plants and grasshoppers we will be able to infer about biomechanical properties that explain why different grasshopper species eat different plant species. It is important to notice that this matrix does not necessarily reflect the interactions that actually happen in the field. In fact, in the field, other traits of both grasshoppers and plants can play a role in the feeding interaction beside the pure biomechanics. But this same approach could be used to analyze bipartide interaction matrices, built based on observed interactions in the field (see below).

We will start by calculating the average leaf trait values of the plants that are consumed by each grasshopper species weighted by the amount of consumed leaves (i.e., CWM). This can be easily done by analytically treating the food preference matrix as it was a community matrix (i.e., considering grasshoppers as "plots" and the amount of consumed leaf of each plant species as "abundances of species").

Let's first have a look on the diet matrix:

```
foodpref[1:6, 1:3]
```

```
##                       Anthoxanthum_odoratum Bromus_erectus Carex_sempervirens
## Anonconotus_alpinus_F                   0.0            0.0                0.7
## Anonconotus_alpinus_M                   1.0            0.0                0.0
## Arcyptera_fusca_F                       2.6           19.9               24.5
## Arcyptera_fusca_M                      10.7            0.0               24.5
## Bohemanella_frigida_F                   0.0            0.0                1.2
## Bohemanella_frigida_M                   0.0            0.0                0.0
```

This matrix is organized with grasshopper species in the rows and plant species in the columns. The numbers represent the amount of leaves of different species (mg) consumed by each grasshopper species during the cafeteria experiment. This same type of data matrix can be obtained by different methods, for instance with field observations for other interaction types like pollination, frugivory and so on. But consider that in the field other traits might also play a role, as phenology for example, beside the pure feeding trait matching between the trophically interacting species.

It is important to note that males and females of the same grasshopper species are presented in distinct rows. This is because sexual dimorphism on morphological traits can have important consequences for food preferences. So males and females of the same species are here considered as different ecological units. This clipping of the food preference matrix shows the amount of leaf of three species eaten by three grasshopper species separately for males "_M" and females "_F". Accordingly, the traits of grasshopper species are also measured separately for males and females as you can see in the grasshopper trait matrix ("grasshopx-traits"). See Chapter 6 of the reference textbook and accompanying R material for more about intraspecific trait variability.

```
head(grasshopxtraits)
```

```
##                           rel.Fi       size
## Anonconotus_alpinus_F 0.3446392  0.7008538
```

```
## Anonconotus_alpinus_M 0.2387203 -0.6278402
## Arcyptera_fusca_F     1.9214781  6.3460157
## Arcyptera_fusca_M     0.9503565  2.9768642
## Bohemanella_frigida_F 0.4647216 -1.2358516
## Bohemanella_frigida_M 0.4807179 -0.6554730
```

This matrix shows two grasshopper traits: size (mg) and incisive strength index (rel.Fi), which is a measurement of biting force.

You can also have a look at the plant trait matrix ("plantxtraits"), where you will find seven leaf traits: work to shear (WS, in J/m), work to punch (WP, in J/m?), thickness (mm), leaf dry matter content (LDMC, in mg/g), Carbon:Nitrogen ratio (CN), leaf nitrogen content (LNC, in mg/g) and specific leaf area (SLA, in mm?/mg). The two first traits represent different aspects of leaf toughness.

```
head(plantxtraits)
```

```
##                           WS      WP Thickness   LDMC    CN   LNC   SLA
## Anthoxanthum_odoratum 0.12  601.91      0.17 240.00 25.09 17.40 26.65
## Bromus_erectus        0.35 1031.21      0.27 366.29 26.46 16.86 14.42
## Carex_sempervirens    0.27 1317.20      0.32 361.61 29.18 15.60 13.18
## Centaurea_uniflora    0.20  394.27      0.43 212.85 18.21 24.50 16.17
## Dactylis_glomerata    0.21  767.52      0.22 298.55 19.66 22.42 20.05
## Festuca_laevigata     0.34  825.48      0.45 376.27 32.57 13.44 11.24
```

Now we just need to follow the steps presented in the R material for computing CWM (R material 5). Below we will use the function `functcomp`, which will do all the work. But we strongly advice for beginners to follow the exercises presented on R material 5 for knowing what the function is actually doing with your data.

For running the function `functcomp`, we need one trait matrix and one community matrix, with rows as sites and columns as species. Our diet matrix has grasshopper species in rows, so it is already prepared for considering the grasshoppers as sites or plots and calculating the CWM of the plants they eat.

```
eatenleaves.cwm <- functcomp(plantxtraits, as.matrix(foodpref))
```

We produced a data frame with the CWM of seven traits for the plants eaten by grasshoppers. We can use these values as a characterization of the feeding preference, in terms of plant traits, of the grasshopper species (i.e. the traits of the plant species most frequently eaten).

```
head(eatenleaves.cwm)
```

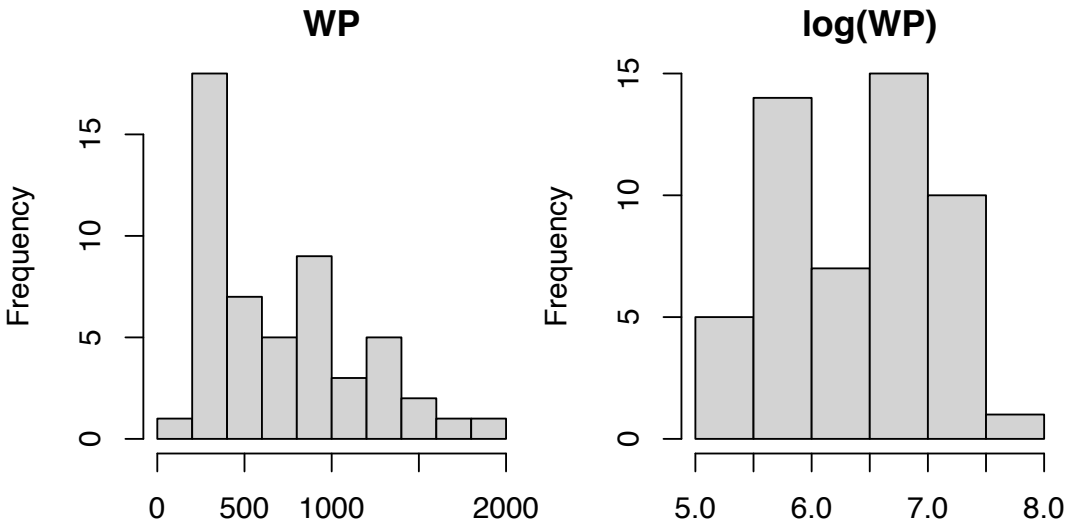
```
##                               WS        WP Thickness     LDMC       CN      LNC
## Anonconotus_alpinus_F 0.08056006  231.8018 0.4725569 181.1967 15.47730 30.02749
## Anonconotus_alpinus_M 0.09054066  238.4880 0.4676085 161.9717 18.24610 26.02377
## Arcyptera_fusca_F     0.29772323 1263.7644 0.3852107 339.7955 29.04666 18.29817
## Arcyptera_fusca_M     0.33868320 1488.7315 0.3768285 364.4289 32.71318 14.68411
## Bohemanella_frigida_F 0.14499272  338.0183 0.4149345 222.3185 18.42063 25.82840
## Bohemanella_frigida_M 0.18930732  346.3488 0.3904618 238.6221 16.74740 28.19285
##                             SLA
## Anonconotus_alpinus_F 16.64724
## Anonconotus_alpinus_M 15.77898
## Arcyptera_fusca_F     13.96656
## Arcyptera_fusca_M     13.01511
## Bohemanella_frigida_F 15.62256
## Bohemanella_frigida_M 15.37130
```

```
summary(eatenleaves.cwm)
```

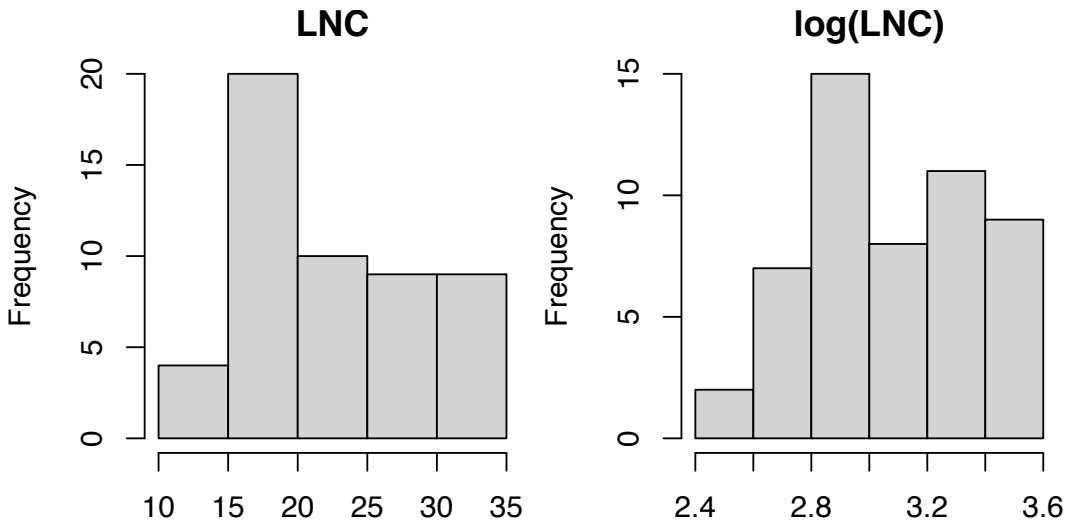
```
##        WS                WP           Thickness          LDMC
##  Min.   :0.07448   Min.   : 194.3   Min.   :0.2341   Min.   :151.4
##  1st Qu.:0.12796   1st Qu.: 313.6   1st Qu.:0.3263   1st Qu.:225.0
##  Median :0.19505   Median : 623.8   Median :0.3759   Median :296.1
##  Mean   :0.20130   Mean   : 707.2   Mean   :0.3695   Mean   :283.2
##  3rd Qu.:0.25251   3rd Qu.: 976.0   3rd Qu.:0.4181   3rd Qu.:340.2
##  Max.   :0.41784   Max.   :1894.3   Max.   :0.4920   Max.   :391.1
##        CN              LNC              SLA
##  Min.   :12.95   Min.   :11.89   Min.   : 9.629
##  1st Qu.:18.23   1st Qu.:17.73   1st Qu.:14.136
##  Median :24.68   Median :20.98   Median :15.916
##  Mean   :23.27   Mean   :22.37   Mean   :16.184
##  3rd Qu.:27.76   3rd Qu.:26.13   3rd Qu.:18.415
##  Max.   :38.24   Max.   :34.80   Max.   :21.766
```

As we can see there is considerable variation on the average leaf trait values of the plants eaten by the different grasshopper species. Let's have a closer look at one mechanical trait and one chemical trait. Let's check if they are more or less normally distributed and if a log transformation can improve normality.

```
par(mfrow = c(1, 2))
par(mar = c(2, 4, 2, 0.5))
hist(eatenleaves.cwm$WP, main = "WP", xlab = "")
hist(log(eatenleaves.cwm$WP), main = "log(WP)", xlab = "")
```




```
par(mfrow = c(1, 2))
par(mar = c(2, 4, 2, 0.5))
hist(eatenleaves.cwm$LNC, main = "LNC", xlab = "")
hist(log(eatenleaves.cwm$LNC), main = "log(LNC)", xlab = "")
```



Ok, it seems that it is better to use log-transformed data in the following analyses.

Now we can test if the variation on average leaf traits eaten by the grasshoppers can be explained by the traits of the grasshoppers. In other words, we can answer to the following question: does grasshopper traits exert any restriction on their food preference? For instance, we can hypothesize that only larger species, with high values of biting force are able to eat tougher leaves. We can test this hypothesis with regression models using CWM of leaf work to punch (WP) as a dependent variable and grasshopper size and incisive strength index as explanatory variables.

```
mod.size <- lm(log(eatenleaves.cwm$WP) ~ grasshopxtraits$size)
summary(mod.size)
```

```
##
## Call:
## lm(formula = log(eatenleaves.cwm$WP) ~ grasshopxtraits$size)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.23253 -0.53462  0.02324  0.46240  1.15397
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            6.37122    0.08780  72.563   <2e-16 ***
## grasshopxtraits$size -0.05338    0.03518  -1.517    0.135
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.633 on 50 degrees of freedom
## Multiple R-squared:  0.04402,    Adjusted R-squared:  0.0249
## F-statistic: 2.302 on 1 and 50 DF,  p-value: 0.1355
```

```
mod.bite <- lm(log(eatenleaves.cwm$WP) ~ grasshopxtraits$rel.Fi)
summary(mod.bite)
```

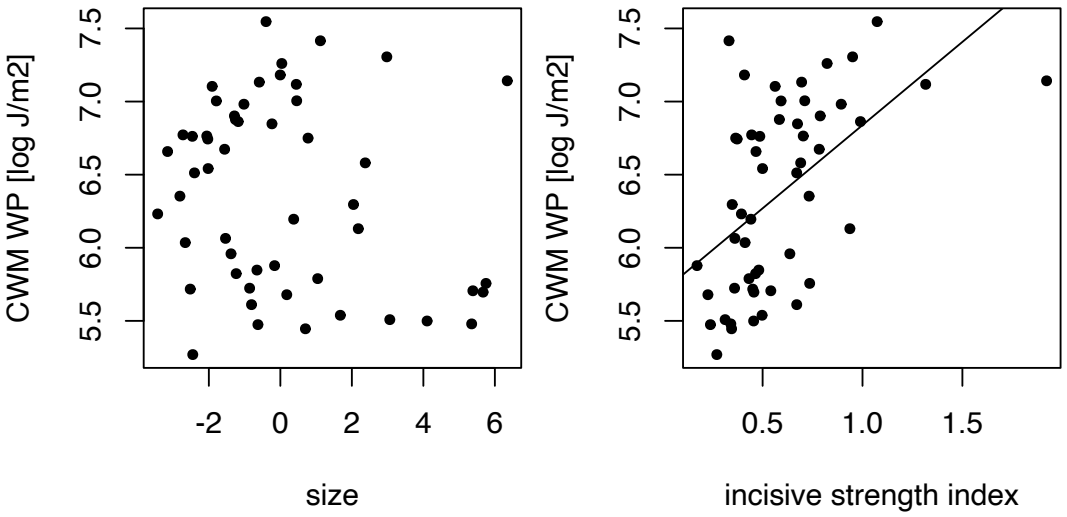
```
##
## Call:
## lm(formula = log(eatenleaves.cwm$WP) ~ grasshopxtraits$rel.Fi)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -0.85293 -0.49655  0.01422  0.49886  1.33762
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                5.7007     0.1671  34.116  < 2e-16 ***
## grasshopxtraits$rel.Fi     1.1381     0.2538   4.484  4.3e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5468 on 50 degrees of freedom
## Multiple R-squared:  0.2868, Adjusted R-squared:  0.2725
## F-statistic:  20.1 on 1 and 50 DF,  p-value: 4.3e-05
```

Very interesting! Body size which is often used as an important surrogate of many aspects of animal functioning does not actually explain food selection by grasshoppers. For explaining the feeding preferences of grasshoppers, we rather need a trait that is more directly related to the ability of biting and chewing. The incisive strength index, which is calculated considering the morphology and size of mouth parts, is able to explain about 27% of the grasshoppers food preference based on leaf toughness (WP).

Let's see how these relationships look like:

```
par(mfrow = c(1, 2))
par(mar = c(4, 4, 1, 0.5))
plot(grasshopxtraits$size, log(eatenleaves.cwm$WP), xlab = "size",
     ylab = "CWM WP [log J/m2]", pch = 20) ## corrigir unidade
plot(grasshopxtraits$rel.Fi, log(eatenleaves.cwm$WP),
     xlab = "incisive strength index", ylab = "CWM WP [log J/m2]",  pch = 20)
abline(mod.bite)
```



Can you now come up with hypotheses related to other functional aspects of the grasshoppers feeding preference? We could think that animals with small body size or with low biting force should have more restricted diet, a more specialized diet, as they can mostly eat soft leaves. Which aspect of the functional feeding preference could reflect the grasshoppers' diet width? Think for a momment…Yes, functional diversity! We can use the function `dbFD` from the *FD* package for calculating metrics of functional diversity. Again, we suggest having a look at the exercises of R material Ch 5 for a better understanding on how this function works. For allowing a better comparison with the results based on CWM, let's calculate functional diversity based on only one trait: work to punch (WP).

```
plantWP <- as.data.frame(plantxtraits$WP) #for calculating FD based on one trait only we nee
row.names(plantWP) <- row.names(plantxtraits) #adding the species names on our new data fram
eatenleaves.FD <- dbFD(plantxtraits, as.matrix(foodpref), message = F,
                       stand.FRic = TRUE)
names(eatenleaves.FD)
```

```
## [1] "nbsp"      "sing.sp"   "FRic"      "qual.FRic" "FEve"      "FDiv"
## [7] "FDis"      "RaoQ"      "CWM"
```

We have calculated several functional diversity indices. Let's have a look on the functional richness (FRic), which is a good representation of niche width. In this case, as we used only one trait, this index will basically return the range on WP values from the leaves eaten by each grasshopper species.

If our hypothesis is right, we should expect a positive relationship of functional richness with body size and biting force.

```
mod.sizeFD <- lm(eatenleaves.FD$FRic ~ grasshopxtraits$size)
summary(mod.sizeFD)
```

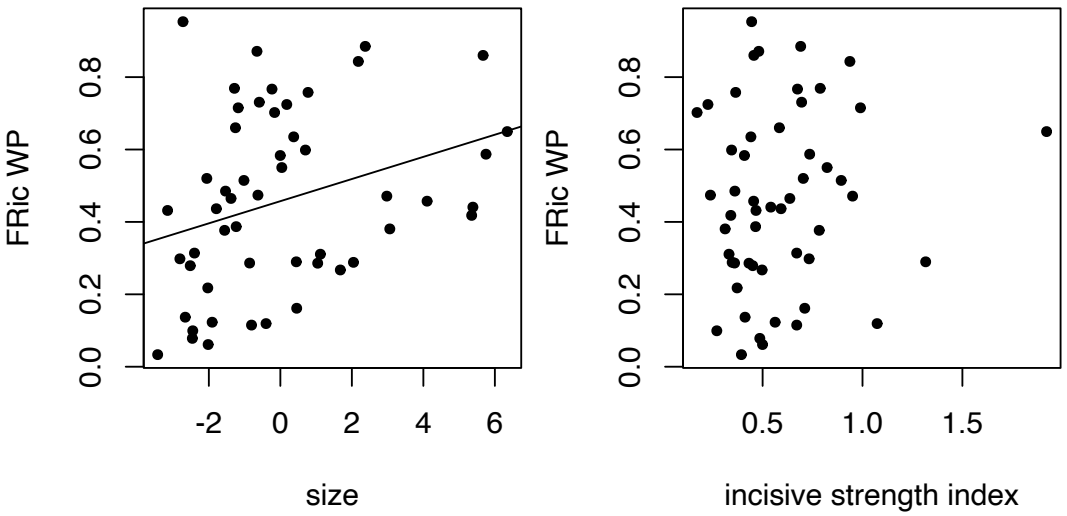
```
##
## Call:
## lm(formula = eatenleaves.FD$FRic ~ grasshopxtraits$size)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33421 -0.18671 -0.03269  0.18203  0.57921
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.45730    0.03290  13.898   <2e-16 ***
## grasshopxtraits$size  0.03061    0.01318   2.321   0.0244 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2372 on 50 degrees of freedom
## Multiple R-squared:  0.0973, Adjusted R-squared:  0.07924
## F-statistic: 5.389 on 1 and 50 DF,  p-value: 0.02438
```

```
mod.biteFD <- lm(eatenleaves.FD$FRic ~ grasshopxtraits$rel.Fi)
summary(mod.biteFD)
```

```
##
## Call:
## lm(formula = eatenleaves.FD$FRic ~ grasshopxtraits$rel.Fi)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40846 -0.16930 -0.01536  0.19208  0.50675
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             0.40686    0.07585   5.364 2.09e-06 ***
## grasshopxtraits$rel.Fi  0.08883    0.11522   0.771    0.444
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.2482 on 50 degrees of freedom
## Multiple R-squared:  0.01175,    Adjusted R-squared:  -0.008019
## F-statistic: 0.5943 on 1 and 50 DF,  p-value: 0.4444
```

```
par(mfrow = c(1, 2))
par(mar = c(4, 4, 1, 0.5))
plot(grasshopxtraits$size, eatenleaves.FD$FRic, xlab = "size", ylab = "FRic WP",
     pch = 20) ## corrigir unidade
abline(mod.sizeFD)
plot(grasshopxtraits$rel.Fi, eatenleaves.FD$FRic, xlab = "incisive strength index",
     ylab = "FRic WP", pch = 20)
```



This time body size was able to explain the diet width based on leaf WP. It is possible to observe that the diet of large animals includes plants representing a larger variation in leaf toughness as compared to small animals. On the other hand, biting force did not show a significant relationship with FRic of WP.

We are done. But, what about if we consider other plant traits for characterizing grasshoppers' food preference? Can you figure out hypotheses about the food preference based in leaf nitrogen content? It is your time to get hands on and explore this data set based on the examples above.

10.4 Testing the response-effect trait framework across trophic levels

The response-effect trait framework is a conceptual tool that can help us to quantify how environmental changes can affect ecosystem functioning via shifts in the community trait composition. As explained in detail in Chapter 10, this chain of indirect effects can also involve different trophic levels. This is because environmental changes can affect trait composition of one trophic level, which, in turn, can result in shifts in trait composition of a second trophic level with consequences to ecosystem processes. For exploring these ideas we will use a data set of grassland sites comprising a gradient of management intensity. Intensification is represented in a scale from 1 to 4 (low to high), reflecting combinations and frequencies of practices like grazing, mowing and fertilization. We also have data on plant and grasshopper traits that can affect plat biomass production. In this data set, these traits are already presented as CWM, that is the community average values weighted by species relative abundance. Leaf dry matter content (LDMC) is a surrogate to leaf density and is

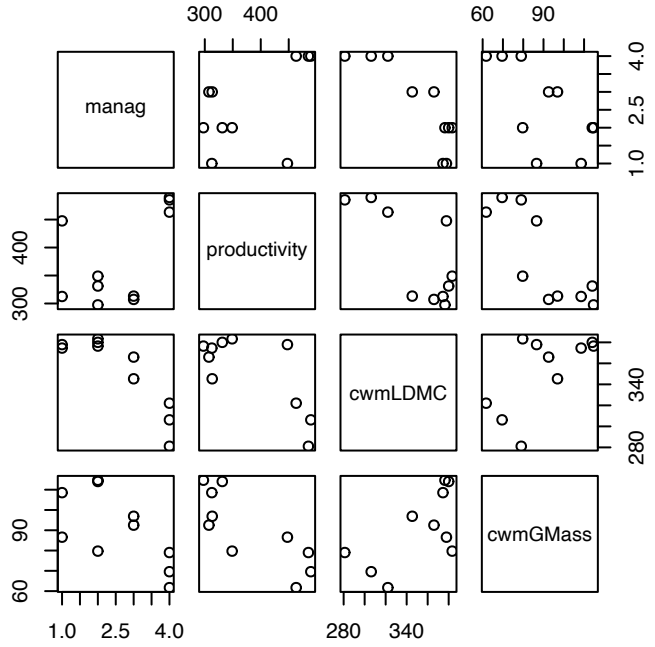
one of the traits related to the Leaf Economics Spectrum (see Chapter 3 in the reference textbook). Grasshopper biomass (GMass) is a measurement of body size. As mentioned in the first part of this exercise, grasshopper body size can be considered as a response trait to leaf biomechanical properties. But we can also expect body size to work as an effect trait if grasshopper body size regulates plant biomass via its consumption. In this exercise, we will use structural equation models (SEM) to test direct and indirect effects of management intensity (manag) on plants and grasshoppers' community functional composition and its consequences to plant biomass production (productivity).

Let's first have a look at the data and the relationship between variables.

```
head(plantxgrasshop)
```

```
##       manag productivity cwmLDMC cwmGMass
## site1     4     485.2041  281.08  79.0204
## site2     4     489.4949  306.22  69.5721
## site3     4     463.2827  322.08  61.7145
## site4     3     313.2150  345.31  96.9029
## site5     3     307.2987  365.97  92.5212
## site6     2     297.5229  376.45 114.6799
```

```
plot(plantxgrasshop)
```



Just by looking at these graphs we can expect to find, at least potentially, many significant relationships between variables. We could start by exploring these potential significant relationships, but here we will follow a hypothesis-testing approach. We will build and test two alternative models representing two alternative explanations (or hypotheses) about the effects of grassland management on productivity mediated (or not) by plant and/or grasshopper traits. Our first model will only consider direct effects of management on productivity. We will also include the effects of plant and grasshopper traits on productivity, but in this model, the functional composition of plants and grasshoppers are not affected by management. We will call it 'model.direct', because it only includes direct effects of different variables on productivity. Our second model includes indirect effects of management on productivity mediated by shifts in plants and grasshoppers functional composition. Because of that, we will call this 'model.indirect'.

Structural equation models (SEM) are graphically represented using path graphs as showed below. Each observed variable is represented by a box and arrows represent the hypothesized causal relationships.

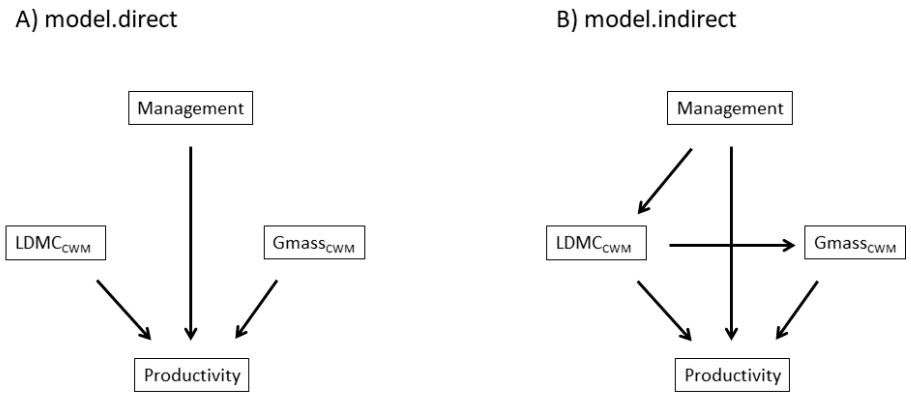


Figure 10.1: This is how our two alternative hypothesized models look like. A) model.direct: management affects productivity only directly. B) model.indirect: management affects productivity both directly and indirectly mediated by shifts in plants and grasshoppers functional composition.You can just draw it in any graphic software of your preference.

The path graph is a graphical representation of our hypothesis, therefore, we need to be able to justify each relationship specified in the model. This is what we call model justification and is a very important step when working with SEM. In our model, we can assume that management intensification represents a higher disturbance frequency and favor species with more acquisitive strategies, which, in turn, should affect productivity. Leaf traits associated to the continuum between acquisitive and conservative resource use strategies, like leaf density (or LDMC) and toughness, can represent biomechanical limitations to chewing herbivores such as grasshoppers. So we can expect that changes in such these traits can affect grasshopper traits like body size and biting force, which, in turn, can affect plant net productivity via consumption.

Now we will use the package *sem* to run both models and test which one better represent the covariance matrix of the observed data. In other words, which model better fit or explain our data and has a higher probability to represent the real set of causal relationships that generated the observed patterns of covariance between our variables. When using the package *sem* you can specify your model in different ways. Here we will use the function `specifyModel`. With this function we can use arrows (->) to represent the hypothesized causal relationships in the same way we did in the path graphs above. Here I named as *beta* the coefficients estimating the effect of one variable on another, analogous to a regression coefficient. As you can see below in the first line of the model, *beta1* specify the effect of management (manag) on productivity, and NA means that I am not fixing any value for this coefficient. It is also necessary to add the estimation of the error of our variables. For this we use double arrows (<->) between one given variable and itself. I named the error estimates as *theta*, but of course you can give any name you want. Once again, NA means that these errors are not fixed, and they will be estimated by maximum likelihood. The exception is the variable management, for which I assume there is no error as we addressed the level of management intensity ourselves.

Note that if you are working with a large model, comprising many variables and relationships between them, it can be more efficient to specify your model using linear equations (see the documentation of the package *sem* for details).

This is how we can specify our model.direct:

```
model.direct <- specifyModel(
  text = "
  manag    ->  productivity,     beta1, NA
  cwmLDMC  ->  productivity,     beta2, NA
  cwmGMass ->  productivity,     beta3, NA
  productivity <-> productivity, theta1, NA
  cwmLDMC      <-> cwmLDMC ,     theta2, NA
  cwmGMass     <-> cwmGMass,     theta3, NA
  manag        <-> manag,        NA, 1
")
```

Now we will run our model using the function `sem` and check the results. We will also calculate the standardized coefficients, which will help us with the interpretation of the model. These coefficients can be understood in the same way of stanrdard coefficients in a multiple regression. Lower values indicate a weak effect and higher values indicate stronger effect.

```
sem.direct <- sem(model.direct, data = plantxgrasshop) # Running the model
summary(sem.direct, fit.indices = c("GFI", "AGFI")) # Checking the results
```

```
##
##  Model Chisquare =  18.2038   Df =  4 Pr(>Chisq) = 0.001125895
##  Goodness-of-fit index =  0.5312083
##  Adjusted goodness-of-fit index =  -0.1719791
##
##  Normalized Residuals
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.0390 -1.2700  0.0000 -0.1173  1.0815  2.5063
##
##  R-square for Endogenous Variables
## productivity
##       0.9145
##
##  Parameter Estimates
##         Estimate    Std Error   z value   Pr(>|z|)
## beta1   -47.484658  10.5119473 -4.517209 6.266020e-06
## beta2    -2.033367   0.2903491 -7.003180 2.502177e-12
## beta3    -2.959877   0.5720389 -5.174259 2.288174e-07
## theta1  994.509323 468.8161910  2.121320 3.389485e-02
## theta2 1310.766271 617.9011459  2.121320 3.389485e-02
## theta3  337.687678 159.1874979  2.121320 3.389485e-02
##
## beta1  productivity <--- manag
## beta2  productivity <--- cwmLDMC
## beta3  productivity <--- cwmGMass
## theta1 productivity <--> productivity
## theta2 cwmLDMC <--> cwmLDMC
## theta3 cwmGMass <--> cwmGMass
##
##  Iterations =  0
```

```
coef(sem.direct, standardized = TRUE) # Asking for the standardized coefficients
```

```
##       beta1       beta2       beta3      theta1      theta2      theta3
## -0.44036776 -0.68271688 -0.50442143  0.08553291  1.00000000  1.00000000
```

First thing you should look at the output of a SEM is the value of Chi square and the fitting indices. The Chi square test the difference between the observed covariance matrix and the covariance matrix generate by your model. In this case, we found a significant difference between these two matrices, which means that there is a very low probability that our hypothesized structure of causal relationships generated the observed covariance matrix. The fitting indices also present very low values.

Let's now run our alternative hypothesized model, where shifts in vegetation traits due to management and its effects on grasshopper traits also determine productivity.

```
model.indirect <- specifyModel(
  text = "
  manag   ->  productivity, beta1, NA
  manag   ->  cwmLDMC,      beta2, NA
  cwmLDMC ->  cwmGMass,     beta4, NA
  cwmLDMC ->  productivity, beta5, NA
  cwmGMass->  productivity, beta6, NA
  productivity <-> productivity, theta1, NA
  cwmLDMC <-> cwmLDMC , theta2, NA
  cwmGMass <-> cwmGMass, theta3, NA
  manag <-> manag, NA, 1
")
sem.indirect <- sem(model.indirect, data = plantxgrasshop) #
summary(sem.indirect, fit.indices = c("GFI", "AGFI"))
```

```
##
##  Model Chisquare =  1.157991   Df =  2 Pr(>Chisq) = 0.5604612
##  Goodness-of-fit index =  0.9418251
##  Adjusted goodness-of-fit index =  0.7091255
##
##  Normalized Residuals
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.836419 -0.405072 -0.001941 -0.011955  0.432089  0.801388
##
##  R-square for Endogenous Variables
## productivity      cwmLDMC     cwmGMass
##       0.8534       0.6793       0.3564
##
##  Parameter Estimates
##        Estimate    Std Error   z value   Pr(>|z|)
## beta1  -47.4846578  18.5625611 -2.558088 1.052496e-02
## beta2  -26.6190323   6.0965410 -4.366252 1.263968e-05
## beta4    0.3251538   0.1456339  2.232679 2.557010e-02
## beta5   -2.0333672   0.6237085 -3.260124 1.113635e-03
## beta6   -2.9598774   0.7449721 -3.973139 7.093167e-05
## theta1 994.5093233 468.8161910  2.121320 3.389485e-02
## theta2 334.5103054 157.6896702  2.121320 3.389485e-02
## theta3 199.1069359  93.8599097  2.121320 3.389485e-02
##
## beta1  productivity <--- manag
## beta2  cwmLDMC <--- manag
## beta4  cwmGMass <--- cwmLDMC
## beta5  productivity <--- cwmLDMC
## beta6  productivity <--- cwmGMass
## theta1 productivity <--> productivity
## theta2 cwmLDMC <--> cwmLDMC
## theta3 cwmGMass <--> cwmGMass
##
##  Iterations =  0
```

```
coef(sem.indirect, standardized = TRUE)
```

```
##      beta1      beta2      beta4      beta5      beta6     theta1     theta2
## -0.5766119 -0.8242003  0.5970316 -0.7974539 -0.6322007  0.1466457  0.3206938
##     theta3
##  0.6435532
```

This seems much better!! There is no significant difference between observed and estimated covariance matrices and the fitting indices show much higher values. Let's place the important statistics in the path graphs representing our alternative models so we can compare them.

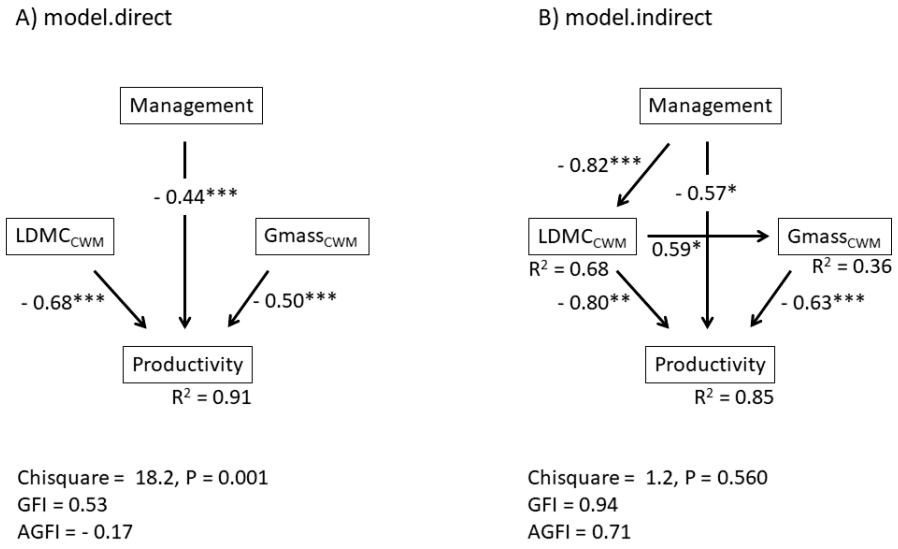


Figure 10.2: This is how we can report the results of our models

As mentioned above, model.direct was rejected, while model.indirect show a reasonable fit to our data. This corroborate the hypothesis that management have both direct and indirect effects (mediated by shifts in plant and herbivore community trait composition) on plant biomass production. Importantly, model.indirect shows that the effects of management on plant biomass production can be both positive and negative. While the direct effect of management intensity on productivity is negative (-0.57), the indirect effect of management intensity on productivity via changes in $LDMC_{CWM}$ is positive (-0.82 x -0.80 = 0.66). In this case, the indirect positive effect is even higher than the negative direct effect. We also can calculate the indirect effect of management intensity via shifts in trait composition of the two trophic levels (-0.82 x 0.59 x -0.63 = 0.30). Thus, we can say that the overall effect of management intensity on productivity is positive, contrary to what we would conclude looking at the direct effect only.

This exercise clearly shows that adding information about traits from different trophic levels can help us understanding the complex effects of environmental changes (like management intensification) on ecosystem processes.

Can you think about other models to test with this data set? Go ahead, give it a shot!

Just one important final note about SEM. This is an exercise, and we are using a very small dataset. Low number of replicates is a limitation when using maximum likelihood (ML) for inferring models as we just did with the package *sem*. Using the d-sep test (Shipley, 2000) can be more appropriated to small data sets and it is already implemented in the package *piecewiseSEM*, but it also has some limitations compared to ML. So it is important

to understand the pros and cons of different methods for running SEM before start working with this powerful tool.

Bibliography

Blomberg, S., Garland, T., and Ives, A. (2003). Testing for phylogenetic signal in comparative data: behavioral traits are more labile. *Evolution; International Journal of Organic Evolution*, 57(4):717–745. PMID: 12778543.

Blonder, B., Lamanna, C., Violle, C., and Enquist, B. (2014). Then-dimensional hypervolume. *Global Ecology and Biogeography*, 23(5):595–609.

Botta-Dukát, Z. (2005). Rao's quadratic entropy as a measure of functional diversity based on multiple traits. *Journal of Vegetation Science*, 16(5):533–540. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1654-1103.2005.tb02393.x.

Botta-Dukát, Z. (2018). The generalized replication principle and the partitioning of functional diversity into independent alpha and beta components. *Ecography*, 41(1):40–50. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ecog.02009.

Cadotte, M. (2017). Functional traits explain ecosystem function through opposing mechanisms. *Ecology Letters*, 20(8):989–996.

Carmona, C., Azcárate, F., de Bello, F., Ollero, H., Lepš, J., and Peco, B. (2012). Taxonomical and functional diversity turnover in mediterranean grasslands: interactions between grazing, habitat type and rainfall. *Journal of Applied Ecology*, 49(5):1084–1093.

Carmona, C., Bello, F., Mason, N., and Lepš, J. (2019). Trait probability density (tpd): measuring functional diversity across scales based on tpd with r. *Ecology*, 100(12).

Carmona, C., de Bello, F., Mason, N., and Lepš, J. (2016). Traits without borders: Integrating functional diversity across scales. *Trends in Ecology & Evolution*, 31(5):382–394.

Carmona, C., Mason, N., Azcárate, F., and Peco, B. (2015a). Inter-annual fluctuations in rainfall shift the functional structure of mediterranean grasslands across gradients of productivity and disturbance. *Journal of Vegetation Science*, 26(3):538–551. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jvs.12260.

Carmona, C., Rota, C., Azcárate, F., and Peco, B. (2015b). More for less: sampling strategies of plant functional traits across local environmental gradients. *Functional Ecology*, 29(4):579–588.

de Bello, F., Botta-Dukát, Z., Lepš, J., and Fibich, P. (2021). Towards a more balanced combination of multiple traits when computing functional differences between species. *Methods in Ecology and Evolution*, n/a(n/a). _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13537.

de Bello, F., Carmona, C., Lepš, J., Szava-Kovats, R., and Pärtel, M. (2016). Functional diversity through the mean trait dissimilarity: resolving shortcomings with existing paradigms and algorithms. *Oecologia*, 180(4):933–940.

de Bello, F., Carmona, C., Mason, N., Sebastià, M., and Lepš, J. (2013). Which trait dissimilarity for functional diversity: trait means or trait overlap? *Journal of Vegetation Science*, 24(5):807–819.

de Bello, F., Lavergne, S., Meynard, C., Lepš, J., and Thuiller, W. (2010a). The partitioning of diversity: showing theseus a way out of the labyrinth. *Journal of Vegetation Science*, 21(5):992–1000.

de Bello, F., Lavorel, S., Albert, C., Thuiller, W., Grigulis, K., Dolezal, J., Janeček, Š., and Lepš, J. (2011). Quantifying the relevance of intraspecific trait variability for functional diversity. *Methods in Ecology and Evolution*, 2(2):163–174. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2041-210X.2010.00071.x.

de Bello, F., Lavorel, S., Díaz, S., Harrington, R., Cornelissen, J., Bardgett, R., Berg, M., Cipriotti, P., Feld, C., Hering, D., Martins da Silva, P., Potts, S., Sandin, L., Sousa, J., Storkey, J., Wardle, D., and Harrison, P. (2010b). Towards an assessment of multiple ecosystem processes and services via functional traits. *Biodiversity and Conservation*, 19(10):2873–2893.

de Bello, F., Lepš, J., and Sebastià, M. (2005). Predictive value of plant traits to grazing along a climatic gradient in the mediterranean. *Journal of Applied Ecology*, 42(5):824–833.

de Bello, F., Lepš, J., and Sebastià, M. (2006). Variations in species and functional plant diversity along climatic and grazing gradients. *Ecography*, 29(6):801–810. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.2006.0906-7590.04683.x.

de Bello, F., Price, J., Münkemüller, T., Liira, J., Zobel, M., Thuiller, W., Gerhold, P., Götzenberger, L., Lavergne, S., Lepš, J., Zobel, K., and Pärtel, M. (2012). Functional species pool framework to test for biotic effects on community assembly. *Ecology*, 93(10):2263–2273. _eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1890/11-1394.1.

de Bello, F., Thuiller, W., Lepš, J., Choler, P., Clément, J., Macek, P., Sebastià, M., and Lavorel, S. (2009). Partitioning of functional diversity reveals the scale and extent of trait convergence and divergence. *Journal of Vegetation Science*, 20(3):475–486.

Dias, A., Berg, M., Bello, F., Oosten, A., Bílá, K., and Moretti, M. (2013). An experimental framework to identify community functional components driving ecosystem processes and services delivery. *Journal of Ecology*, 101(1):29–37. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2745.12024.

Diaz, S., Lavorel, S., de Bello, F., Quetier, F., Grigulis, K., and Robson, T. (2007). Incorporating plant functional diversity effects in ecosystem service assessments. *Proceedings of the National Academy of Sciences*, 104(52):20684–20689.

Durka, W. and Michalski, S. (2012). Daphne: a dated phylogeny of a large european flora for phylogenetically informed ecological analyses. *Ecology*, 93(10):2297–2297.

Fontana, S., Petchey, O., and Pomati, F. (2016). Individual-level trait diversity concepts and indices to comprehensively describe community change in multidimensional trait space. *Functional Ecology*, 30(5):808–818. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2435.12551.

Galland, T., Adeux, G., Dvořáková, H., E-Vojtkó, A., Orbán, I., Lussu, M., Puy, J., Blažek, P., Lanta, V., Lepš, J., Bello, F., Carmona, C., Valencia, E., and Götzenberger, L. (2019). Colonization resistance and establishment success along gradients of functional and phylogenetic diversity in experimental plant communities. *Journal of Ecology*, 107(5):2090–2104. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2745.13246.

Garnier, E., Lavorel, S., Ansquer, P., Castro, H., Cruz, P., Dolezal, J., Eriksson, O., Fortunel, C., Freitas, H., Golodets, C., Grigulis, K., Jouany, C., Kazakou, E., Kigel, J., Kleyer, M., Lehsten, V., Lepš, J., Meier, T., Pakeman, R., Papadimitriou, M., Papanastasis, V., Quested, H., Quétier, F., Robson, M., Roumet, C., Rusch, G., Skarpe, C., Sternberg, M., Theau, J., Thébault, A., Vile, D., and Zarovali, M. (2007). Assessing the effects of land-use change on plant traits, communities and ecosystem functioning in grasslands: A standardized methodology and lessons from an application to 11 european sites. *Annals of Botany*, 99(5):967–985.

Geange, S., Pledger, S., Burns, K., and Shima, J. (2011). A unified analysis of niche overlap incorporating data of different types. *Methods in Ecology and Evolution*, 2(2):175–184. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2041-210X.2010.00070.x.

Gotelli, N. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, 81(9):2606–2621. _eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1890/0012-9658

Gotelli, N. and Graves, G. (1996). *Null Models in Ecology*. Accepted: 2009-07-21T18:33:00Z.

Götzenberger, L., Botta-Dukát, Z., Lepš, J., Pärtel, M., Zobel, M., and de Bello, F. (2016). Which randomizations detect convergence and divergence in trait-based community assembly? a test of commonly used null models. *Journal of Vegetation Science*, 27(6):1275–1287.

Götzenberger, L., de Bello, F., Bråthen, K., Davison, J., Dubuis, A., Guisan, A., Lepš, J., Lindborg, R., Moora, M., Pärtel, M., Pellissier, L., Pottier, J., Vittoz, P., Zobel, K., and Zobel, M. (2012). Ecological assembly rules in plant communities—approaches, patterns and prospects. *Biological Reviews*, 87(1):111–127.

Hanisch, M., Schweiger, O., Cord, A., Volk, M., and Knapp, S. (2020). Plant functional traits shape multiple ecosystem services, their trade-offs and synergies in grasslands. *Journal of Applied Ecology*, 57(8):1535–1550. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2664.13644.

Hardy, O. (2008). Testing the spatial phylogenetic structure of local communities: statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology*, 96(5):914–926.

Hardy, O. and Jost, L. (2008). Interpreting and estimating measures of community phylogenetic structuring. *Journal of Ecology*, 96(5):849–852.

Ibanez, S., Lavorel, S., Puijalon, S., and Moretti, M. (2013). Herbivory mediated by coupling between biomechanical traits of plants and grasshoppers. *Functional Ecology*, 27(2):479–489. _eprint: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/1365-2435.12058.

Jost, L. (2007). Partitioning diversity into independent alpha and beta components. *Ecology*, 88(10):2427–2439.

Kleyer, M., Dray, S., Bello, F., Lepš, J., Pakeman, R., Strauss, B., Thuiller, W., and Lavorel, S. (2012). Assessing species and community functional responses to environmental gradients: which multivariate methods? *Journal of Vegetation Science*, 23(5):805–821. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1654-1103.2012.01402.x.

Laliberté, E. and Legendre, P. (2010). A distance-based framework for measuring functional diversity from multiple traits. *Ecology*, 91(1):299–305. _eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1890/08-2244.1.

Lavorel, S., Grigulis, K., McIntyre, S., Williams, N., Garden, D., Dorrough, J., Berman, S., Quétier, F., Thébault, A., and Bonis, A. (2007). Assessing functional diversity in the field – methodology matters! *Functional Ecology*, 22(1):134–147.

Lepš, J., Bello, F., Šmilauer, P., and Doležal, J. (2011). Community trait response to environment: disentangling species turnover vs intraspecific trait variability effects. *Ecography*, 34(5):856–863. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1600-0587.2010.06904.x.

Lepš, J., De Bello, F., Lavorel, S., and Berman, S. (2006). Quantifying and interpreting functional diversity of natural communities: practical considerations matter. *Preslia*, 78(4):481–501.

Loreau, M. and Hector, A. (2001). Partitioning selection and complementarity in biodiversity experiments. *Nature*, 412(6842):72–76.

Mason, N., Bello, F., Mouillot, D., Pavoine, S., and Dray, S. (2013). A guide for using functional diversity indices to reveal changes in assembly processes along ecological gradients. *Journal of Vegetation Science*, 24(5):794–806. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jvs.12013.

Mason, N., Mouillot, D., Lee, W., and Wilson, J. (2005). Functional richness, functional evenness and functional divergence: the primary components of functional diversity. *Oikos*, 111(1):112–118.

Messier, J., McGill, B., and Lechowicz, M. (2010). How do traits vary across ecological scales? a case for trait-based ecology. *Ecology Letters*, 13(7):838–848. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1461-0248.2010.01476.x.

Moretti, M., de Bello, F., Ibanez, S., Fontana, S., Pezzatti, G., Dziock, F., Rixen, C., and Lavorel, S. (2013). Linking traits between plants and invertebrate herbivores to track functional effects of land-use changes. *Journal of Vegetation Science*, 24(5):949–962.

Mouchet, M., Guilhaumon, F., Villéger, S., Mason, N., Tomasini, J., and Mouillot, D. (2008). Towards a consensus for calculating dendrogram-based functional diversity indices. *Oikos*, 117(5):794–800.

Mouillot, D., Mason, W., Dumay, O., and Wilson, J. (2005). Functional regularity: a neglected aspect of functional diversity. *Oecologia*, 142(3):353–359.

Májeková, M., Paal, T., Plowman, N., Bryndová, M., Kasari, L., Norberg, A., Weiss, M., Bishop, T., Luke, S., Sam, K., Bagousse-Pinguet, Y., Lepš, J., Götzenberger, L., and Bello, F. (2016). Evaluating functional diversity: Missing trait data and the importance of species abundance structure and data transformation. *PLOS ONE*, 11(2):e0149270. Publisher: Public Library of Science.

Pagel, M. (1999). Inferring the historical patterns of biological evolution. *Nature*, 401(6756):877–884. Number: 6756 Publisher: Nature Publishing Group.

Pakeman, R. and Quested, H. (2007). Sampling plant functional traits: What proportion of the species need to be measured? *Applied Vegetation Science*, 10(1):91–96. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1654-109X.2007.tb00507.x.

Pavoine, S. and Bonsall, M. (2011). Measuring biodiversity to explain community assembly: a unified approach. *Biological Reviews*, 86(4):792–812. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-185X.2010.00171.x.

Pavoine, S., Vallet, J., Dufour, A., Gachet, S., and Daniel, H. (2009). On the challenge of treating various types of variables: application for improving the measurement of functional diversity. *Oikos*, 118(3):391–402. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1600-0706.2008.16668.x.

Petchey, O. and Gaston, K. (2002). Functional diversity (fd), species richness and community composition. *Ecology Letters*, 5(3):402–411. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1046/j.1461-0248.2002.00339.x.

Petchey, O. and Gaston, K. (2006). Functional diversity: back to basics and looking forward. *Ecology Letters*, 9(6):741–758. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1461-0248.2006.00924.x.

Pistón, N., Bello, F., Dias, A., Götzenberger, L., Rosado, B., Mattos, E., Salguero-Gómez, R., and Carmona, C. (2019). Multidimensional ecological analyses demonstrate how interactions between functional traits shape fitness and life history strategies. *Journal of Ecology*, 107(5):2317–2328.

Pärtel, M., Szava-Kovats, R., and Zobel, M. (2011). Dark diversity: shedding light on absent species. *Trends in Ecology & Evolution*, 26(3):124–128.

Qian, H. and Jin, Y. (2016). An updated megaphylogeny of plants, a tool for generating plant phylogenies and an analysis of phylogenetic community structure. *Journal of Plant Ecology*, 9(2):233–239.

Ricotta, C., de Bello, F., Moretti, M., Caccianiga, M., Cerabolini, B., and Pavoine, S. (2016). Measuring the functional redundancy of biological communities: a quantitative guide. *Methods in Ecology and Evolution*, 7(11):1386–1395.

Ricotta, C. and Moretti, M. (2011). Cwm and rao's quadratic diversity: a unified framework for functional ecology. *Oecologia*, 167(1):181–188.

Shipley, B. (2000). A new inferential test for path models based on directed acyclic graphs. *Structural Equation Modeling: A Multidisciplinary Journal*, 7(2):206–218.

Siefert, A., Violle, C., Chalmandrier, L., Albert, C., Taudiere, A., Fajardo, A., Aarssen, L., Baraloto, C., Carlucci, M., Cianciaruso, M., de L. Dantas, V., de Bello, F., Duarte, L., Fonseca, C., Freschet, G., Gaucherand, S., Gross, N., Hikosaka, K., Jackson, B., Jung, V., Kamiyama, C., Katabuchi, M., Kembel, S., Kichenin, E., Kraft, N., Lagerström, A., Bagousse-Pinguet, Y., Li, Y., Mason, N., Messier, J., Nakashizuka, T., Overton, J., Peltzer, D., Pérez-Ramos, I., Pillar, V., Prentice, H., Richardson, S., Sasaki, T., Schamp, B., Schöb, C., Shipley, B., Sundqvist, M., Sykes, M., Vandewalle, M., and Wardle, D. (2015). A global meta-analysis of the relative extent of intraspecific trait variation in plant communities. *Ecology Letters*, 18(12):1406–1419.

Smith, S. and Brown, J. (2018). Constructing a broadly inclusive seed plant phylogeny. *American Journal of Botany*, 105(3):302–314.

Swanson, H., Lysy, M., Power, M., Stasko, A., Johnson, J., and Reist, J. (2015). A new probabilistic method for quantifyingn-dimensional ecological niches and niche overlap. *Ecology*, 96(2):318–324.

Swenson, N. (2014). *Functional and Phylogenetic Ecology in R*. Springer New York.

Villéger, S., Mason, N., and Mouillot, D. (2008). New multidimensional functional diversity indices for a multifaceted framework in functional ecology. *Ecology*, 89(8):2290–2301.

Villéger, S. and Mouillot, D. (2008). Additive partitioning of diversity including species differences: a comment on hardy & senterre (2007). *Journal of Ecology*, 96(5):845–848.

Zanne, A., Tank, D., Cornwell, W., Eastman, J., Smith, S., FitzJohn, R., McGlinn, D., O'Meara, B., Moles, A., Reich, P., Royer, D., Soltis, D., Stevens, P., Westoby, M., Wright, I., Aarssen, L., Bertin, R., Calaminus, A., Govaerts, R., Hemmings, F., Leishman, M., Oleksyn, J., Soltis, P., Swenson, N., Warman, L., and Beaulieu, J. (2013). Three keys to the radiation of angiosperms into freezing environments. *Nature*, 506(7486):89–92.

Zelený, D. (2018). Which results of the standard test for community-weighted mean approach are too optimistic? *Journal of Vegetation Science*, 29(6):953–966.