

Applied Multivariate Statistics in R

Applied Multivariate Statistics in R

JONATHAN D. BAKKER

UNIVERSITY OF WASHINGTON
SEATTLE, WA



Applied Multivariate Statistics in R Copyright © 2024 by Jonathan D. Bakker is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License, except where otherwise noted.

Contents

Acknowledgements	vii
Introduction	1

Part I. Foundational Concepts

1. Installing and Running Software	3
2. Reproducible Research	11
3. Loading Data	18
4. R Basics	23
5. Data Adjustments	41
6. Transformations	51
7. Relativizations	55
8. Matrix Algebra Basics	67
9. Matrix Algebra to Solve a Linear Regression	76
10. Eigenanalysis	81
11. Properties of Distance Measures	85
12. Common Distance Measures	92
13. Multivariate Outlier Analysis	109

Part II. Group Comparisons

14. ANOVA / MANOVA	116
15. Sample Datasets	119
16. Permutation Tests	122
17. ANOSIM	130
18. Mantel Test	140
19. MRPP	147
20. PERMANOVA	153
21. PERMDISP	166
22. RRPP	176
23. Complex Models	188
24. Controlling Permutations	201

25. Restricting Permutations	207
26. Comparison of Techniques	231

Part III. Classification

27. Types of Cluster Analyses	239
28. Hierarchical Cluster Analysis	242
29. k-Means Cluster Analysis	263
30. Using Groups	270
31. Discriminant Analysis	281
32. Overview of Classification and Regression Trees	299
33. Univariate Regression Trees	305
34. Multivariate Regression Trees	334

Part IV. Ordinations (Data Reduction and Visualization)

35. Types of Ordination Methods	352
36. PCA	355
37. NMDS	380
38. PCoA	401
39. RDA and dbRDA	408
40. CA, DCA, and CCA	418
41. Comparison of Ordination Techniques	434
42. General Graphing Principles	445
43. Visualizing and Interpreting Ordinations	467

Part V. Follow-Up Tests

44. SIMPER	496
45. ISA	505
46. TITAN	522

Appendix 1: Order of Data Adjustments	536
---------------------------------------	-----

Appendix 2: Structure of Complex Experimental Designs	538
---	-----

Appendix 4: Contrasts	545
-----------------------	-----

Acknowledgements

I thank Bruce McCune for permission to use the Garry oak plant community dataset that is the primary example throughout these notes. Other datasets are drawn from published resources as noted where used.

All data are available here: <https://github.com/jon-bakker/appliedmultivariatestatistics>

Please send comments and suggestions for how this can be improved to jbakker@uw.edu.

Introduction

This book is a work in progress. It began as notes for students in SEFS 502, *Analytical Techniques in Community Ecology*, at the University of Washington. These notes provide a background in multivariate analysis of a wide range of data. My emphasis is on the practical application of these techniques.

Many students have used this course, and their associated project, as part of their thesis or dissertation. A number of students have also published their work in the peer-reviewed literature.

Most of the functions used here are provided within R packages as indicated at the start of each chapter. Other data files, scripts, and functions are available through a GitHub repository associated with this book (<https://github.com/jon-bakker/appliedmultivariatestatistics>).

The book is organized in five parts:

- Foundational Concepts – an introduction to R, data adjustments, matrix algebra, and distance measures
- Group Comparisons – ways to test for differences between pre-defined groups
- Classification – ways to explore the data and group similar observations together
- Ordinations (Data Visualization) – ways to reduce the dimensionality of multivariate data and/or to visualize patterns within it
- Follow-Up Tests – tests to identify how individual response variables relate to one or more explanatory variables

PART I

FOUNDATIONAL CONCEPTS

This section explores concepts that are necessary for all other aspects of the subject. Concepts are presented in short chapters so that the reader can more easily find the information that they seek.

Introduction to R

- Installing and Running Software
- Loading Data
- Reproducible Research
- R Basics

Data Adjustments

- Data Adjustments
- Transformations
- Relativizations
- Order of Data Adjustments

Matrix Algebra

- Matrix Algebra Basics
- Matrix Algebra to Solve a Linear Regression
- Eigenanalysis

Distance Measures

- Properties of Distance Measures
- Common Distance Measures
- Multivariate Outlier Analysis

1. Installing and Running Software

Learning Objectives

- To demonstrate how to install R and RStudio software.
- To become familiar with ways of navigating in R.
- To learn how to install, load, and update R packages.
- To understand how R help files are structured.

Resources

RStudio cheat sheets for reference:

- RStudio IDE
- Base R
- Cheat Sheet::VEGAN

Introduction

R is an open source statistical language and is extremely versatile and customizable. The base installation of R includes some capabilities for writing scripts and taking other actions, but RStudio is a more versatile graphical user interface (GUI) through which R can be accessed.

Installing and Running R

The official R website is: <http://cran.r-project.org/>. However, there are multiple mirrors that contain the exact same information. You can choose the mirror that is nearest your location. Go to the R website (or a mirror), and choose the appropriate operating system from the top center of the page (box titled 'Download and Install R').

- Windows: Click on 'base', and then on the setup program (currently **R-4.3.2-win.exe**). Save this file to a working directory.
- Mac: Choose the setup program. Save this file to a working directory.

Once the file has downloaded, double-click on it to run it. Accept all defaults to install it on your hard drive.

If you would like to install R on a thumb drive (with enough room!), change the install destination during the installation, and choose NOT to create a Start Menu folder or desktop icon. Accept all other defaults. (Note: I haven't fully tested this; you may need to install software to create a virtual PC on your thumb drive first). For more details, see Appendix A of Dalgaard (2008) or Torfs & Brauer (2014).

By default, R operates from a **command line interface**. When you open R, the main window is displayed. This window, called the 'R Console', is where commands are executed, and their numerical/textual results displayed.

R also uses several other windows:

- Graphics
- Editor
- Data editor

These windows are only displayed when necessary. The Editor window is where multiple lines of commands can be written and saved. This permits you to execute a series of commands sequentially. It is extremely useful for saving code so it can easily be rerun in the future.

Installing and Running RStudio

While the standard installation of R does not include a graphical user interface (GUI), several are available as packages that can be installed. In this course, we will not be using a GUI as I want you to see and understand what is happening throughout the analyses. However, the default setting of R is not entirely satisfactory either. Instead, we will use **RStudio**, an IDE ('integrated development environment') for R that is available as a free download.

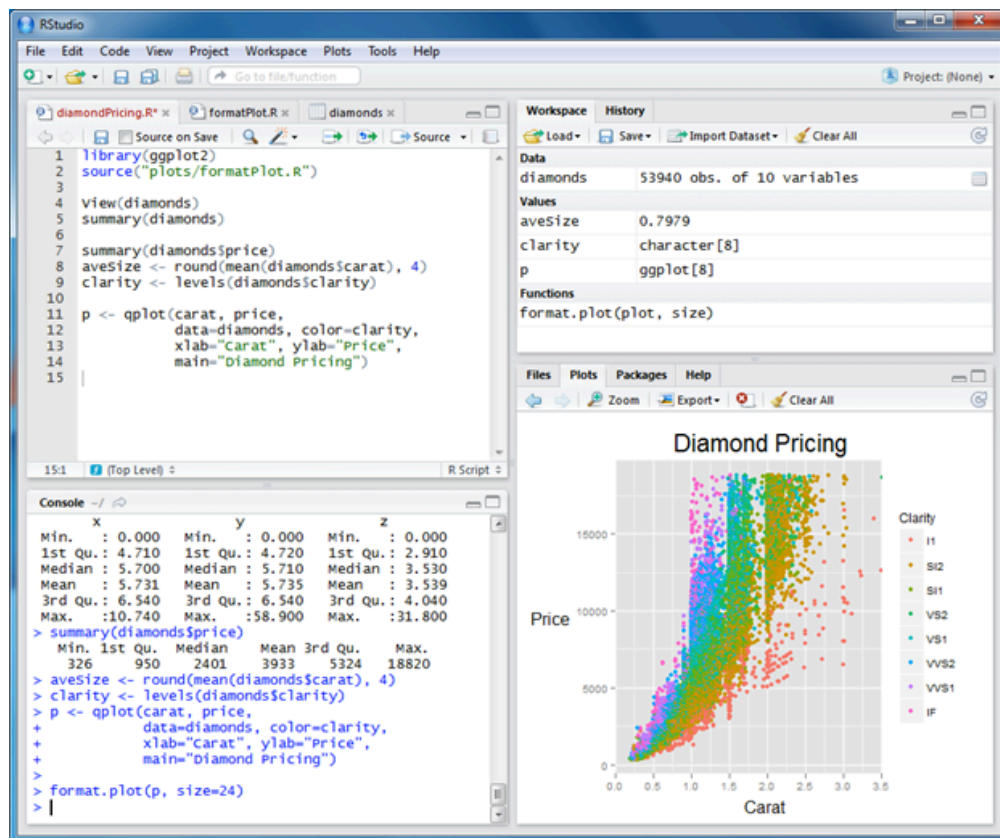
RStudio is available from <https://posit.co/download/rstudio-desktop/>. The current version is **2023.09.1+494**.

A guide to RStudio is provided on the RStudio IDE cheat sheet. More detail is available in the book by van der Loo & de Jonge (2012).

When you open RStudio, R is automatically opened as well. The version that is running is displayed in the Console. If you have multiple versions of R installed, you can select the version that you want to use. To do so, go to 'Tools / Global Options' and select the 'Change...' button below R Version. A list of the installed versions will be displayed; select the one you desire. You may have to restart RStudio for this to take effect.

RStudio Basics

The RStudio window is divided into several panes as shown and described below.



Example of the RStudio window showing its four panes: console (lower left), editor (upper left), workspace and history (upper right), and files, plots, packages, and help (lower right).

In the lower left is the **R Console** – the same as you see when you open R directly. There are also tabs here for Terminal and Background Jobs.

In the upper left is an **editor** pane (this may not be displayed if you have not opened or created a script). The editor pane is where you can open, edit, and save a script file containing many lines of code. This is analogous to the editor window in R, but has additional features such as:

- **Multiple scripts** can be open simultaneously, each appearing as a tab in this pane.
- **Font color** changes to distinguish functions, comments, etc.
- **Auto-complete:** when you type a left parenthesis or bracket, the closing parenthesis or bracket is automatically created.
- **Quotation marks:** if you need to place material in quotation marks, you can do so by highlighting it and then typing the quotation mark – they are automatically added before and after the selected text.
- **Error-checking:** when you place the cursor after a parenthesis, the corresponding parenthesis that closes the clause is highlighted. This is particularly helpful when you have multiple functions nested one inside another.
- Can **comment** or uncomment multiple lines simultaneously (Edit tab: Comment/Uncomment Lines' command).

In the upper right pane are four tabs:

- **Environment:** displays summary details of the objects you have created in R (more on those later).

- **History:** a list of all of the commands that you've run during this session, in order.
- **Connections:** allows you to connect to different data sources.
- **Tutorial:** allows you to run tutorials about RStudio.

Finally, the lower right pane contains six tabs:

- **Files:** allows simple navigation within your directory structure.
- **Plots:** displays the graphics that you produce through commands sent to the Console. Can move left to right to move between figures.
- **Packages:** allows you to load or remove packages by simply selecting the checkbox beside their name. Packages that are not installed (see below) are not included in this list.
- **Help:** access to the help files associated with the base version of R and with installed packages.
- **Viewer:** to view local web content.
- **Presentation:** ability to create and display presentations that include R code and output.

Navigating in R

The '>' prompt in the R Console indicates that R is waiting for a command.

You can execute commands from RStudio's editor pane by clicking 'Run' or pressing the Ctrl-Enter key combination. Selecting code before executing will alter what is run:

- Place the cursor anywhere in a line of code (note: does not have to be at the end of the line). 'Run' will execute that line of code. If the line is part of a set of commands, the full set will be run. However, this is not true if the commands are embedded within other types of instructions. For example, a function consists of code 'wrapped' by commands about argument names, etc. Placing the cursor in a line of code within a function will run the code but not the associated wrapper lines.
- Highlight multiple lines of code. 'Run' will execute all of the highlighted code.
- Highlight a portion of a line. 'Run' will execute just the highlighted text and not the rest of the line. This can be helpful when debugging code, for example as it allows you to execute one portion at a time.

Once you've entered a command, you can use the up arrow to scroll through commands you've already run. You can then edit and rerun a command without having to retype it all.

A '+' prompt means that R is waiting for you to finish a command. This is often intentional – such as when a command spans multiple lines – but also occurs if you forgot to enter the closing parenthesis in a command.

R Packages

A **function** is a piece of code that conducts a specific action. The base installation of R contains numerous functions for actions such as manipulating, summarizing, or graphing data. Some of the commonly used functions are summarized on reference cards – these are helpful resources to keep at hand.

One of the appealing features of R is that it is open-source and therefore users can create their own functions. When users create multiple functions that are relevant to a particular theme, they are encouraged to share those functions with others by bundling them as a **package** and posting that package on the R website (<https://cran.r-project.org/web/packages/>), where they are available for anyone to use. There are many more packages available on the R website than are evident in the

base installation of R: more than 20,000 as of December 2023! New packages are being constantly created by users, and many are actively maintained and updated by those who created them or others who are interested in that theme.

Installing and Updating Packages

Packages can be downloaded from the website, and are one of the main ways to customize your usage of R. Packages that we will commonly use in this course include:

- `vegan`
- `labdsv`
- `permute`
- `plyr`
- `rpart`
- `tidyverse` (single name for multiple packages, including `ggplot2`)

Most of these packages are being actively revised and updated. Others get built upon in other packages – for example, there are numerous packages that provide extensions to `ggplot2`.

For example, go to the R website and find the link to the `vegan` package (I recommend learning how to navigate to it, but here's the direct link: <https://cran.r-project.org/web/packages/vegan/index.html>). The current version is **2.6-4**. The website contains the source code in Windows and Mac formats, a PDF reference manual, and several vignettes (i.e., worked examples). You can install this package by choosing the appropriate binary file from the website and extracting this file into a folder in the R library.

However, an easier way to install and update packages is from within R or RStudio. From the 'Packages' tab in the lower right pane of RStudio, simply click on the 'Install' button and follow the directions in the various windows. Notice that you can install several packages simultaneously. Some packages use functions from other packages; these other packages (dependencies) are automatically installed by default. If you look back at the R Console, you'll see the commands that were executed throughout this process.

To update packages that have already been installed, click the 'Update' button. All packages are compared against the versions on the website, and a list of packages with updated versions is displayed in a window. When you select the package(s) to update and choose 'ok', the new versions of these packages are downloaded and installed.

Loading R Packages

Simply installing a package doesn't make it accessible to you – you must **load** it into R before you can use it. If you haven't loaded a package, R doesn't 'know' it exists. One reason for this is because different packages sometimes use the same function name to do different things. If both packages are loaded, R won't know which function you mean.

To manually load a package in RStudio, simply go to the 'Packages' tab in the lower right pane, find the package you want, and select the checkbox beside the package name. The command that is executed by doing so is displayed in the R Console.

To load a package in R, or in a script, use the `library()` function:

```
library(vegan)
```

If the package requires other packages, those other packages are automatically loaded as well. If the package is not installed, this function returns an error. A related function, `require()`, also loads

packages but differs in that it returns an error if unable to do so. This can be helpful if you want a script not to execute if the package is not available.

Unloading a package can be done using the `detach()` function:

```
detach("package:vegan", unload = TRUE)
```

In RStudio, you can also unload a package by deselecting the checkbox beside the package name.

Help!

Finding Help

Details about R can be found in many places:

- Books (see resources listed on course website)
- R reference cards (e.g., Baggott 2012) and cheatsheets such as those from RStudio
- Help files associated with the program
- *The R Journal*, the open access, refereed journal of the R project for statistical computing
- Internet searches. I often include 'cran' at the beginning of a query to focus attention on R-related content. Much good information is available through sites such as <https://stackoverflow.com/>.

You can get help about a function or topic in several ways. The 'Help' tab in the lower right pane of RStudio contains links to help files for the base version of R and for all installed packages. It also includes a search engine so that you can find help (note that it autocompletes with names of functions from loaded packages that meet your criterion).

If you are searching for a function contained within a package that has already been loaded, and you know the name of the function, you can search for it directly from the R Console:

```
help(plot)
?plot      # ? is equivalent to help()
```

The help files that are displayed are the same as those available if you navigated there from the 'Help' tab or searched the PDF or html reference manuals available through the R website.

If you aren't sure of the name of the function or topic, you can do a more general search of the loaded packages. To search for a function name that includes specified text:

```
apropos("plot")
```

Use the `help.search()` function to search all of the packages installed on your machine, even those that aren't loaded. For example, soon we'll use the `decostand()` function to standardize data. This function is available in the `vegan` package.

```
?decostand # Doesn't work
help.search("decostand") # Works (if package is installed)!
```

The resulting window returns a list of packages and functions (in the format `package::function`) that contain the specified text. This information can be entered together to open the specified help file even if the package hasn't been loaded:

```
help(decostand, package = vegan)
?vegan::decostand # Equivalent but faster entry
```

Of course, if you load the `vegan` package first, you can obtain the help file for `decostand()` directly:

```
library(vegan)
?decostand
```

Finally, you can use the `args()` command to display a list of the arguments, with their default values, in the R Console:
`args(decostand)`

Interpreting Help Files

R help files are organized in a standard format, though not all files have all sections. Understanding this format will help you interpret the contents. Learning how to skim a help file to identify the key bit that you want to tweak is a valuable skill to develop when using R.

Section	Contents
Description	A generic description of what a function does. Can be hard to interpret as it uses specific R language. An example of how the command would be entered on the command line. The standard format is something like: <code>command(arg1, arg2, ...)</code>
Usage	In this case, <code>command</code> is the name of the command, and <code>arg1</code> and <code>arg2</code> are the arguments that enable you to customize the action. The <code>'...'</code> indicates that additional arguments can be included.
Arguments	The details about each argument. Some are required, others are not. Some have default values, others do not. If there are a limited number of possible values, that set of values are usually identified here.
Details	Further explanation of how the function works, how it differs from other functions, etc. Sometimes the possible values for an argument are explained here.
Value	Type of object returned (not always applicable). Object may contain multiple elements that can be extracted individually, manipulated, etc.
Warning	
Note	General comments about function.
References	Articles or books that describe the function (or that the function is based upon).
Author(s)	Person / people who wrote this function.
See Also	A list of other functions that will perform similar or related actions.
Examples	Text that can be copied and pasted directly into R to see how the function works. Often, the correct answer is described in a comment (text following a <code>#</code> symbol).

R help files are organized in a standard format, though not all files have all sections. Understanding this format will help you interpret the contents. Learning how to skim a help file to identify the key bit that you want to tweak is a valuable skill to develop when using R.

An easy way to see R's potential as graphical software is through a demonstration:

`demo(graphics)`

While this demonstration runs, the commands to produce a given graph are shown in the Console and the graph itself is displayed in the 'Plots' tab. In this case, the code is actually a little above the command prompt in the Console because the code for the next graph has been executed but the results are not displayed until you advance to the next graph. The net result, however, is that, when you find a graph that you like, you can copy the code and modify it for your data.

Another demo:

`demo(persp)`

Good books highlighting the graphical capabilities of R include Murrell (2006), Sarkar (2008), Wickham (2009), Chang (2013), and Wickham & Grolemund (2017). The more recent of these titles focus on the `ggplot2` package, which provides extremely simple yet powerful graphing capabilities. I highly recommend exploring it. Although it was not used to create the above demonstrations, we'll use it later. The help files for this package (<https://ggplot2.tidyverse.org/reference/>) include many visual examples.

References

- Baggott, M. 2012. *R reference card 2.0*. 6 pg. <http://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf>
- Chang, W. 2013. *R graphics cookbook*. O'Reilly, Sebastopol, CA.
- Dalgaard, P. 2008. *Introductory statistics with R*. Second edition (First edition, 2002). Springer, New York, NY.
- Murrell, P. 2006. *R graphics*. Chapman & Hall/CRC, Boca Raton, FL.
- Sarkar, D. 2008. *Lattice: multivariate data visualization with R*. Springer, New York, NY.
- Torfs, P., and C. Brauer. 2014. *A (very) short introduction to R*. <http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf>
- van der Loo, M.P.J., and E. de Jonge. 2012. *Learning RStudio for R statistical computing*. Packt, Birmingham, UK.
- Wickham, H. 2009. *ggplot2: elegant graphics for data analysis*. Springer, New York, NY.
- Wickham, H., and G. Grolemund. 2017. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly, Sebastopol, CA. <http://r4ds.had.co.nz/>

Media Attributions

- RStudio

2. Reproducible Research

Learning Objectives

- To consider the importance of reproducible research, and how R can enhance this.
- To identify principles for analysis organization, scripting, and workflow design.

Opening Comments

There is a growing concern in science about the extent to which it is reproducible. In biomedical studies, for example, the strain of organism or conditions in the lab can have much larger effects than were previously appreciated (e.g., Lithgow et al. 2018). In ecology, these concerns recently led to the formation of the Society for Open, Reliable, and Transparent Ecology and Evolutionary Biology (SORTEE).

A similar issue relates computationally. An individual analysis requires hundreds of large and small decisions, and some of those decisions can dramatically affect the conclusions. Computational reproducibility should be easier to demonstrate than reproducibility in lab environments. Kitzes et al. (2018) state that “a research project is computationally reproducible if a second investigator (including you in the future) can recreate the final reported results of the project, including key quantitative findings, tables, and figures, given only a set of files and written instructions.” Ideally, the script you produce for this course will fit this description! Filazolla & Lortie (2022) provide recommendations for writing clean code.

The field of data science has much to teach us about computational reproducibility. Options even exist to analyze R code directly (McGowan et al. 2020).

General Practices for Reproducible Research

Clearly separate, label, and document all data, files, and operations that occur on data and files. In other words, **organize** files in a clear directory structure and prepare **metadata** that describe them.

Document all operations fully, automating them as much as possible, and avoiding manual intervention in the workflow when feasible. In other words, write **scripts** that perform each step automatically. Where this is not possible, document all manual steps needed to complete a task.

Design a workflow as a sequence of small steps that are glued together, with intermediate outputs from one step feeding into the next step as inputs. In other words, prepare your overall **workflow design** so that you understand the different operations that need to occur, the sequence they need to occur in, and how they support one another.

Each of these practices is briefly described below. See Kitzes et al. (2018) for more information.

Analysis Organization

Projects

RStudio uses **projects** to help organize analyses. Specifically, a project is self-contained and provides access to data, scripts, etc. We will follow this approach. For additional details, see Robinson (2016) and Bryan (2017).

The main directory for the project is the working directory. This is where R will automatically look for files, and where files that are produced will automatically be saved. This also provides portability, as the project file will be useable by a collaborator even if their directory structure differs from yours.

When you open a R project, the working directory is automatically set as the folder in which the project file is stored. You can confirm the working directory using the `getwd()` function:

Note: one advantage of using a R project is that the working directory is automatically set. If needed, however, you can use the `setwd()` function to change the working directory. You can specify a particular folder by typing out the address, or you can use the `choose.dir()` function to open a window through which you can navigate to the desired folder.

```
setwd("C://Users/jbakker/Desktop/SEFS 502/")  
setwd(choose.dir())
```

Note that the slashes that map out the folder hierarchy are the opposite of what is conventionally used in Windows.

Create a folder in which you will store all files associated with SEFS 502 class notes and examples. Once you've created this folder, save a R project within it. Use this project to open R and work through class examples.

Sub-folders

A key advantage of setting a working directory is that it allows a project to have a stable starting point yet to remain organized by having items in the appropriate sub-folders. Commonly used sub-folders include:

- data
- graphics
- images
- library
- notes
- output
- scripts (note: most data scientists now would recommend that you store your scripts on GitHub. See 'Workflow design' below for more details)

From the working directory, you can navigate using the **relative path** of a file rather than its absolute path. Sub-folders are referenced by simply including them in the name of the file you are accessing. For example, if the file 'data.csv' was stored in the data sub-folder, we could open it like this:

```
dataa <- read.csv("data/data.csv", header = TRUE, row.names = 1)
```

Using relative paths keeps your code portable among machines and operating systems (Cooper & Hsing 2017). However, note that you do need to have the same sub-folders in your project folder.

In the folder that contains your SEFS 502 class project, create the following sub-folders:

- **data**
- **scripts**
- **functions**
- **graphics**

You can also create additional sub-folders, such as one for class notes, one for readings, etc. if desired.

Project Options

RStudio allows you to save your history (record of code that was run previously) and the objects that were created earlier. However, I advise against this because these shortcuts increase the chance of errors creeping in. For example, objects created during earlier sessions are still available for manipulation even if the current version of the script does not create them.

To turn these options off in RStudio, go to the 'Tools' menu and select 'Project Options'. Under the 'General' settings, set these options to 'No':

- **Restore .RData into workspace at startup**
- **Save workspace to .RData on exit**
- **Always save history (even if not saving .RData)**

Scripting

R Scripts

Scripting is one of the powerful aspects of R that distinguishes it from point-and-click statistical software. When a script is clearly written, it is easy to re-run an analysis as desired – for example, after a data entry error is fixed or if you decide to focus on a particular subset of the data. To capitalize on this ability, it is necessary to be able to manipulate your data in R. New users often want to export their data to Excel, manipulate it there, and re-import it into R. **Resist this urge.** Working in R helps you avoid errors that can creep in in Excel – such as calculating an average over the wrong range of cells – and that are extremely difficult to detect (Broman & Woo 2018).

A script is simply a text file containing the code used (actions taken), along with comments clarifying why those actions were taken. I save these files with a '.R' suffix to distinguish them from other text files. Scripts can be created within RStudio's editor pane (my preference) but could also be created in R's Editor or even in a simple text editor like WordPad. Commands have to be copied and pasted into R for execution.

A script can include many different elements, including:

- Data import
- Define functions that will be used elsewhere (later) in the script
- Data adjustments
- Analysis
- Graphing

These elements can be organized in several ways. In RStudio, for example, you can organize your code into **sections**. Sections can be inserted via the 'Code -> Insert Section' command, or you adding a comment that includes at least four trailing dashes (—), equal signs (====), or pound signs (####).

You can use the 'Jump to' menu at the bottom of the script editor to navigate between sections.

Individual sections can be 'folded' (minimized) to make the overall structure more apparent. The code within that section is replaced with an icon consisting of a blue box with a bidirectional arrow in it. This is also helpful because you can select and run the entire folded section simply by highlighting the symbol that icon.

If your script includes multiple sections:

- Alt-O minimizes all sections
- Alt-Shift-O maximizes all sections

Interactive Notebooks (Jupyter, RMarkdown)

Interactive notebooks provide the capability to create a single document containing code, its output (statistical results, figures, tables, etc.), and narrative text. They have been getting a lot of attention recently. Key advantages are that the connections between data and output are explicit, and the output is automatically updated if the data change.

There are two broad types of interactive notebooks:

- Project Jupyter creates a file that contains the code, output, and narrative text. This file is not of publication quality but is helpful for sharing an analysis with collaborators. UW is testing opportunities to use JupyterHub for teaching; see the link in Canvas if you want to explore or use this. Please note that this course's JupyterHub workspace is only active for this quarter, so if you use it you will need to download any files that you wish to keep at the end of the quarter.
- RMarkdown: A RMarkdown file consists of code and narrative text; Tierney (2020) provides a nice overview of RMarkdown and how to use it. The code can include instructions to create figures or tables. When this file is processed, the output specified by the code is produced and the resulting document is saved as a .pdf or .html file. As such, this approach is closely aligned with publication – in fact, Borcard et al. (2018) state that their entire book was written using RMarkdown in RStudio! (I considered creating these notes in RMarkdown but apparently it doesn't integrate with the PressBooks publishing platform).

Workflow Design

Ecologists (myself included) are rarely trained as programmers, but knowing some of the 'best practices' that programmers use can improve the quality of our scripts. Examples of these practices include:

- **Modularizing code** (i.e., writing custom functions) rather than copying and pasting it to apply it to different objects. This prevents errors where you update the code in one location in your script but forget to update it elsewhere. Modularizing code is also helpful for maintaining

consistency. For example, imagine that you want to create multiple figures with a consistent format. If the script is written as a function, it can be called multiple times with different data. Formatting details would remain the same from figure to figure. And, if you decided to adjust the formatting you would only have to do so in one place!

- Verifying that code is working correctly via **assertions** and **unit tests** (automated testing of units of code)
- **Version control** to keep track of changes. Many (most?) data scientists use Git for this purpose. Closely associated with this is GitHub, which enables code sharing. The data files for this book are available in the course's GitHub repository (<https://github.com/jon-bakker/appliedmultivariatestatistics>).

Cooper & Hsing (2017) is an excellent resource in this regard. There is also a growing body of open-source resources (free on-line lessons; in-person workshops at various locations) available through Software Carpentry. And, the 'Happy Git with R' website appears to be quite readable and useful.

Some best practices and guidelines for formatting code are provided in Table 1 below.

TABLE 1 Common best practices and guidelines for formatting code in biology. Some of these practices are language dependent and must be adjusted accordingly

Formatting	Description	Best practices	Examples
1. Horizontal length	The number of characters within a line	Between 80 and 120 characters with longer sections separated by returns	Functions with many arguments can have a return after each argument
2. Horizontal ordering	The order in which functions are executed	Nonmanipulative operations should occur before others	(1) Data loaded (2) Columns added (3) Aggregation
3. Horizontal spacing	The distance between characters separated by spaces	Space should be added between objects, arguments and operators	<code>data = datasetName</code> <code>list = c(Obj1, Obj2)</code>
4. Indenting	A starting position away from the margin	Indenting indicates nestedness of lines within a larger element	Used in functions, if/else statements and loops
5. Naming conventions	The text pattern for naming objects, functions or classes	Consistency is paramount. Consider different patterns for objects versus functions	<code>snake_case</code> <code>camelCase</code> <code>Kebab-Case</code>
6. Parentheses	Different bracket types to indicate closure around operations	Add horizontal or vertical spacing when appropriate to make obvious open/close parentheses. Try to minimize nesting parentheses	<code>()</code> <code>[]</code> <code>{}</code>
7. Vertical length	The total number of lines associated with a script	Script length is case-by-case but ideally less is more	A 1,000-line script could be split into multiple scripts (e.g. functions vs. analysis)
8. Vertical ordering	The order in which lines are presented in the script	Dependencies should be listed before dependents	(1) Libraries (2) Functions (3) Loading data (4) Analysis
9. Vertical spacing	The distance between lines separated by returns (i.e. lines)	Similar things should be grouped to indicate relatedness. The more lines between, the more unrelated	A linear model with all its diagnostic functions would be listed together

Common best practices and guidelines for formatting code. From Filazolla & Lortie (2022).

McCune & Grace (2002; ch. 8) provide a flowchart and tabular techniques that can assist in this process. See this Appendix for a summary. Workflows can be described verbally or graphically, as in this example:

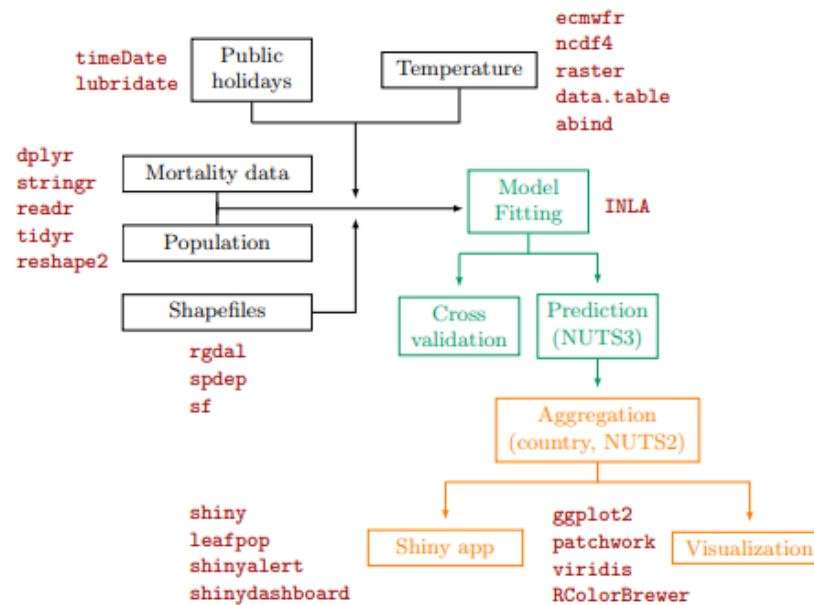


Figure 1: Diagram of the workflow: data wrangling in black, analysis in green, post-processing in orange and packages in red (timeDate; Wuertz et al., 2023, lubridate; Grolemund and Wickham, 2011, dplyr; Wickham et al., 2023a, stringr; Wickham, 2022, readr; Wickham et al., 2022, tidyr; Wickham et al., 2023b, reshape2; Wickham, 2007, rgdal; Bivand et al., 2023, spdep; Bivand, 2022, sf; Pebesma, 2018, shiny; Chang et al., 2022, leafpop; Appelhans and Detsch, 2021, shinyalert; Attali and Edwards, 2021, shinydashboard; Chang and Borges Ribeiro, 2021, ecmwf; Hufkens et al., 2019, ncd4; Pierce, 2023, raster; Hijmans (2023), data.table; Dowle and Srinivasan, 2022, abind; Plate and Heiberger, 2016, ggplot2; Wickham, 2016, patchwork; Pedersen, 2022, viridis; Garnier et al., 2021, RColorBrewer; Neuwirth, 2022). NUTS stands for Nomenclature of Territorial Units for Statistics with NUTS3 being the highest spatial resolution available and NUTS2 coarser but appropriate for policy making.

Example workflow, showing packages used during data wrangling, analysis, and post-processing. From Konstantinoudis et al. (2023).

Conclusions

The scientific process is built on the idea of reproducibility, both in the field and at the computer. The ability to script an analysis is essential to ensuring that analyses can be shared and repeated.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Broman, K.W., and K.H. Woo. 2018. Data organization in spreadsheets. *The American Statistician* 72(1):2-10.
- Bryan, J. 2017. *Project-oriented workflow*. <https://www.tidyverse.org/blog/2017/12/workflow-vs-script/>
- Cooper, N., and P-Y. Hsing. 2017. *A guide to reproducible code in ecology and evolution*. British Ecological Society, London, UK. <http://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf>

Filazzola, A., and C.J. Lortie. 2022. A call for clean code to effectively communicate science. *Methods in Ecology and Evolution* 13:2119-2128.

Kitzes, J., D. Turek, and F. Deniz (eds.). 2018. *The practice of reproducible research: case studies and lessons from the data-intensive sciences*. University of California, Oakland, CA. <http://www.practicereproducibleresearch.org/>

Konstantinoudis, G., V. Gómez-Rubio, M. Cameletti, M. Pirani, G. Baio, and M. Blangiardo. 2023. A workflow for estimating and visualising excess mortality during the COVID-19 pandemic. *The R Journal* 15(2):89-104.

Lithgow, G.J., M. Driscoll, and P. Phillips. 2018. A long journey to reproducible results. *Nature* 548:387-388.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

McGowan, L.D., S. Kross, and J. Leek. 2020. Tools for analyzing R code the tidy way. *The R Journal* 12:226-242.

Robinson, A. 2016. *icebreakR*. <https://cran.rstudio.com/doc/contrib/Robinson-icebreaker.pdf>

Tierney, N. 2020. RMarkdown for scientists. <https://rmd4sci.njtierney.com/>

Media Attributions

- Filazolla.Lortie.2022.Table1
- Konstantinoudis.et.al.2023_Table1 © Konstantinoudis et al. (2023) is licensed under a CC BY (Attribution) license

3. Loading Data

Learning Objectives

- To demonstrate how to load data in R.
- To introduce the sample datasets that we will use throughout the quarter.
- To introduce how to use matching (`%in%`) to index an object and create new objects.

Resources

- Broman & Woo (2018)
- RStudio cheat sheets for reference:
- Data Import with `readr`, `readxl`, and `googlesheets4`

Introduction

There are several ways to load data into R. Small datasets can be typed directly and assigned to an object, though this is not practical for larger datasets. My preferred method for loading data is to organize it in Excel, save the file, and then read it into R. Note that these functions assume your data are organized in a rectangular format, following the recommendations of Broman & Woo (2018).

Key Takeaways

Data files should be organized in rectangular format. The simplest form has rows as samples and columns as data:

- Each **row** is a unique record, such as the observations from one plot at one time.
- Each **column** is a variable. Some of these may be explanatory (e.g., a column identifying the plot and a column identifying the time of each sample was taken) and others may be responses (columns identifying different species in a community, or other measurements made in that plot and time).
- Each **cell** is the value of a given variable (column) for a given sample (row). Missing values should

be indicated as 'NA' or as zero.

While you can read data into R by typing in the full path to the desired file, it is preferred to establish a project folder and work therein as discussed in the 'Reproducible Research' chapter. When you open a R project file, the working directory is automatically set to the folder in which it resides.

You can hard-code the name of the file that you want to load. For example, if the file is saved in the 'data' sub-folder and is named 'data.csv':

```
dataa <- read.csv("data/data.csv")
```

(Note: I don't recommend using 'data' as an object name as there is a `data()` function within R)

Alternatively, you could use `file.choose()` to navigate to and select the desired file:

```
dataa <- read.csv(file.choose())
```

There are times when I find this helpful because it can be quick, but note that this is not automated – if you ran the script again, it would pause here until you selected the file.

Text files are the most common type of files to be loaded, but others are possible as well.

Text Files

The main function to load data is called `read.table()`. This is a generic function with arguments that allow you to customize the call to reflect how your data file is formatted. Some standard file formats have had the required arguments hard-coded: `read.delim()` for tab-delimited text files, and `read.csv()` for CSV files. The arguments that I find most helpful include:

- `file` – the key required argument; the name of the file to be read.
- `header` – whether the names of the variables are included in the first line of the file. The default is that they are not (`header = FALSE`), but I almost always include them in this fashion and therefore set `header = TRUE`.
- `row.names` – the name or number of a column containing a field that you want to assign to row names. Note that this will not work if the referenced column does not contain a unique record for each row.
- `na.strings` – unique values in the dataset that you want to be replaced with 'NA' (Not Available). For example, missing data might have been coded as '999'; you obviously would not want to include these values when calculating an average.

As an example, let's load the oak plant community dataset that is introduced below. This file has column names in the first row, so we will include the `header` argument. It also has a unique code for each entry in the first column, so we will include the `row.names` argument:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
```

The `readr` package, part of the `tidyverse`, provides another set of functions to read rectangular data.

Other File Types

The `readxl` package allows you to load data from a specific sheet of a Microsoft Excel file. Formulae within the Excel file will not be kept; just the resulting value will be loaded.

Large datasets are often stored in relational databases. There are packages that allow you to build and run queries in these databases, and to export the data into R. `RODBC` is one example of this type of package; many others are listed here:

<https://cran.r-project.org/web/views/Databases.html>

Verifying That Data Are Loaded Correctly

Loading data is (perhaps obviously) only useful if you assign the result to an object – otherwise, all you’ve done is displayed it in the Console.

It’s important to verify that the data were loaded correctly. There are several ways to do so by examining the resulting object:

- Check the size or dimensions of the object. This information is reported in the ‘Environment’ panel (upper-right quadrant) of RStudio, but we can also display it in the Console using the `dim()` function. The results are always ordered as the number of rows followed by the number of columns.
- View the first few records using `head()`. Can you guess what the `tail()` function does?
- View a data summary using `summary()`.
- View the structure of the object using `str()`.

Sample Dataset: Oak Plant Communities

The primary dataset that we’ll use as an example throughout the course is from *Quercus garryana* (aka Garry oak, Oregon white oak) stands in the Willamette Valley (Thilenius 1963, 1968). These data were included as an example with PC-ORD, and I have reformatted them for R – I thank Bruce McCune for granting permission to use them!

Three files are associated with this dataset. These and all other data files are available through this book’s GitHub site (<https://github.com/jon-bakker/appliedmultivariatestatistics>):

- Oak_Metadata.docx
- Oak_data_47x216.csv
- Oak_species_189x5.csv

Download these files and save them in the ‘data’ sub-folder within your SEFS 502 folder (the one that contains the R project for these classes). We’ll use them throughout the quarter.

Oak_Metadata.docx explains the other files. We will not be loading it into R, but be sure to look through it. You will get to create a similar metadata file for your class project.

Oak_data_47x216.csv contains the response and explanatory variables for each of the 47 stands. The response variables are the abundances of 189 species in each stand. The explanatory variables are 27 stand-level attributes. I call them explanatory variables for simplicity; in reality, it would be more accurate to term them ‘potential explanatory variables’. Rows correspond to plots and columns

correspond to variables in this object. Most analyses assume the data are structured this way. If your data are reversed, you can re-organize them this way using the `t()` function to transpose the object (however, if they are reversed you may also have issues with regard to the classes of the objects).

Oak_species_189x5.csv contains the species codes associated with this project along with some simple information about each taxon, including its scientific name and life form (tree, shrub, herb, graminoid).

Note: An alternate way to organize these data is to have the response and explanatory data in separate files. Having separate files is helpful for keeping the two datasets distinct, but can cause problems if, for example, the order of data is rearranged in one but not the other. As a result, **the preferred approach is to store the species data and explanatory data together in a single file that is loaded into R and then manipulated there to create new objects containing the desired components.**

Loading and Indexing the Oak Plant Community Data

We are going to be using these data throughout the quarter. Each time we do so, we will need to load them and make some initial data adjustments. We begin by loading both data files:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
```

One of the columns in `Oak_species` is a list of the species codes. We can select the response variables from `Oak` by matching the column names from `Oak` against the names in this list. Doing so is much easier and less error-prone than manually typing them all out. We will use an extremely useful operator, `%in%`, to do so. It works like the `match()` function, but is more intuitive: it selects those elements from the object on its left that are in the object on its right, and ignores elements that are not present in both objects.

```
Oak_abund <- Oak[, colnames(Oak) %in% Oak_species$SpeciesCode]
```

Use the functions that were introduced above (Verifying That Data Are Loaded Correctly) to explore this new object and satisfy yourself that it contains just the species data.

We can also use the `%in%` operator to find non-matches (i.e., column names that are not included in the list of species codes):

```
Oak_explan <- Oak[, ! colnames(Oak) %in% Oak_species$SpeciesCode]
```

Be sure you understand what happened here! These variables should all be explanatory variables (i.e., stand-level attributes). However, if a species was not named identically in both objects then it would also be included in `Oak_explan`.

Key Takeaways

If data are being combined across objects, it is much safer to do so via matching than by assuming that the samples are in the same order in both objects.

References

Broman, K.W., and K.H. Woo. 2018. Data organization in spreadsheets. *The American Statistician* 72:2-10. doi: 10.1080/0031305.2017.1375989

Thilenius, J.F. 1963. *Synecology of the white-oak (Quercus garryana Douglas) woodlands of the Willamette Valley, Oregon*. Ph.D. dissertation. Department of Botany and Plant Pathology, Oregon State University, Corvallis, OR. 151 p.

Thilenius, J.F. 1968. The *Quercus garryana* forests of the Willamette Valley, Oregon. *Ecology* 49:1124-1133.

4. R Basics

Learning Objectives

- To understand the object-oriented nature of R, including object names and classes.
- To understand how objects can be manipulated via operators and indexing.
- To introduce ways to combine functions via nested functions and piping.
- To introduce the tidyverse and other ways to manipulate data, including combining objects.

Resources

Manly & Navarro Alberto (2017, ch. 1)

RStudio cheat sheets for reference:

- RStudio IDE
- Base R
- Data Import with `readr`, `readxl`, and `googlesheets4`
- Apply Functions with `purrr`
- Data Tidying with `tidyr`
- Data Transformation with `dplyr`
- Data Visualization with `ggplot2`
- Dates and times with `lubridate`
- Factors with `forcats`
- String Manipulation with `stringr`
- Cheat Sheet::VEGAN
- Git and GitHub with RStudio
- Other cheat sheets available at <https://posit.co/resources/cheatsheets/>

Key Packages

`tidyverse`, `plyr`

Introduction

If you are new to R, you might find it helpful to read the brief “Field Guide to the R Ecosystem” (Sellors 2019), which introduces the main components of the R ‘ecosystem’. Robinson (2016) also summarizes the key aspects of R in his “icebreakerR”. If you are familiar with SAS or SPSS, you may find the ‘Quick-R’ website particularly helpful.

For an introduction to the basic instructions about R, see the appendix to chapter 1 of Manly & Navarro Alberto (2017), chapter 1 of Dalgaard (2008), Paradis (2005; sections 2-3; online), and Torfs & Brauer (2014). Somewhat more advanced references include Adler (2010) and Borcard et al. (2018). Finally, very comprehensive reference materials are provided in the official R manual by Venables et al. (2023) and in many other publications, including Crawley (2012), Wickham (2014), and Wickham & Grolemund (2017).

For day-to-day reminders when using R, cheatsheets such as those linked to above can be very helpful.

Assigning Names

By convention, R code is displayed in a `different font` than regular text.

Name objects by **assigning** them to a name:

```
name <- object
```

R is Object Oriented

R is **object oriented**. This means that each variable, dataset, function, etc. is stored as an object. Each object is named, and can be manipulated.

Objects are created by **assigning** them to a name. When naming objects, there are a few conventions to keep in mind:

- Must begin with a letter (A-Z, a-z)
- Can include letters, digits (0-9), dots (.), and underscores (_)
- Are **case-sensitive** – `n` and `N` are different objects
- No size limits that I know of. You can create long and descriptive names for your objects ... but you'll have to type them out every time you need to call them.
- If you specify an object that already exists, the existing value assigned to that name is overwritten without warning

Although there is considerable flexibility in how objects are named, consistency is very helpful. Some key recommendations are provided in the textbox below.

Naming Conventions

The Tidyverse Style Guide (Wickham 2020) recommends:

- Use only lowercase letters, numbers, and the underscore (`_`) when naming variables and functions.
- Use the underscore to separate words within a name. For example, `day_one` rather than `dayone`.
- Use nouns for variable names and verbs for function names.
- Avoid the names of common functions and variables. For example, if you named your data object as `data` it would easily be confused with the `data()` function.

Operators

Objects are manipulated via operators and functions (other objects). Common operators include:

Operator	Description
Mathematical	
<code>+</code> <code>-</code>	Addition, Subtraction
<code>*</code> <code>/</code>	Multiplication, Division
<code>^</code>	Exponentiation
Relational	
<code><-</code> <code>=</code>	Assignment: items to right of operator are assigned to name on left of operator. While these operators are synonymous, ' <code><-</code> ' is recommended because it can only be used in assignment while ' <code>=</code> ' can be used in many other situations. <code>?assignOps</code> for details.
<code>-></code>	Rightwards assignment: items to left of operator are assigned to name on right of operator
<code>==</code>	Test of equality
<code>!=</code>	Not equal to
<code><</code>	Less than
<code><=</code>	Less than or equal to
<code>></code>	Greater than
<code>>=</code>	Greater than or equal to
Logical	
<code>&</code> <code>&&</code>	And
<code> </code> <code> </code>	Or
<code>!</code>	Not

?Syntax for details about many of these operators, including their precedence (the order in which they are evaluated).

If you type in an expression but don't assign it to a name, the result will be displayed on the screen but will not be stored as an object in memory. For example:

```
2 + 3 # Result not stored in memory
```

```
n <- 2 + 3
```

```
n
```

```
n + 3
```

Do you see the difference? This is a trivial example because it involves a single number, but the principle holds even with complex analyses (e.g., of objects that consist of many elements). Here is a random sample of 100 observations from a standard normal distribution (i.e., with mean = 0 and standard deviation = 1):

```
N <- rnorm(100)
```

Now that all of these elements have been assigned to the object `N`, they can be manipulated *en masse*. To multiply each of these random samples by 2:

```
2 * N
```

How would we save these new numbers to an object?

We could also calculate the average of these random samples:

```
mean(N)
```

Your mean value should not match mine exactly. Why not?

Multiple commands can be written on a single line if separated by semi-colons. This can be helpful to save space in a program script. For example, we assigned the result of adding 2 and 3 to the object `n` above, but this result wasn't displayed on the screen until we requested it. We could do both functions on a single line, separated by a semi-colon:

```
n <- (4 * 5) / 2; n # Note that we've now reassigned 'n' to a new object!
```

We can also wrap a function in parentheses to simultaneously execute it and display it on the screen:

```
(m <- 1:10)
```

Comments

Functions can be annotated by adding comments after a `#`. Anything from the `#` to the end of the line is ignored by R.

Comments are useful for annotating code, such as explaining what a line does or what output is produced. For example, I often use a comment to document the dimensionality of an object – this gives me a quick way to check that the code is producing an object of the correct size when I rerun it. However, some argue that comments should be used minimally as they easily become out-of-date and can therefore be confusing when checking code (Filazzola & Lortie 2022).

Comments can also be helpful when writing larger pieces of code, as you can selectively turn lines of code off or on by adding or removing a `#` from the front of the line. In RStudio, you can apply this to multiple lines at once by highlighting those lines and selecting 'Code -> Comment/Uncomment Lines'.

In the RStudio editor pane, comments are displayed in a different color than non-commented code.

Object Classes

Every object in R belongs to one or more **classes**. Classes have distinct **attributes** which control what type(s) of data an object can store, and how those objects may be used. Many functions are built to only work for objects of a specified class. The help file for a function will often specify which class(es) of object it works on.

Class	Description
numeric	One-dimensional. 'Regular' numbers (e.g., 4.2).
integer	One-dimensional. Whole numbers (e.g., 4).
character	One-dimensional. Text (e.g., four).
matrix	Two-dimensional. All data within an array must be of the same type.
array	Multi-dimensional. All data within an array must be of the same type.
data.frame	Two-dimensional; resembles a matrix, but data within different columns (vectors) may be of different types. Data within a given column must be of the same type, and all columns must be of the same length. This is one of the most flexible and common ways of storing data.
list	Multi-dimensional; similar to a data frame but the 'columns' do not have to be the same length. The objects that result from R functions are often stored as lists.
factor	A collection of items. Can be character (forest, prairie) or numerical (1, 2, 3). If specified as explanatory variables in an analysis, character data are often recognized as factors and treated appropriately, but numerical data may be assumed to be continuous and treated as covariates unless specified as a factor. If the levels are ordinal (e.g., low, medium, high), the correct ordering can be specified.
logical	Binary (True/False). Often used to index data by identifying data that meet specified criteria.

A **vector** is a one-dimensional series of values, all of which must be of the same class (e.g., integer, numeric, character, logical). Another way to think about this is that the vector will be assigned to the class that applies to all values. For example, use the `class()` function to find out the class of each of these objects:

```
x <- c(1:4)
y <- c(1:3, 4.2)
z <- c(1:3, "four")
```

You can also ask whether an object is of a particular class:

```
is.character(x)
is.character(z)
# Can replace the class name with that of other classes
```

You can force an object into a desired class using a function such as `as.matrix()`; R will do so if possible or return an error message.

A key aspect of importing data into R is verifying that the data have been assigned to the proper classes so that they are handled appropriately during analyses. If data are not assigned to the

proper class, the results of an analysis may be very different than expected. For example, consider an explanatory variable with numerical values corresponding to levels of a factor (1 = urban, 2 = suburban, and 3 = rural). Unless instructed otherwise, R will assume during data import that this variable is numeric. If this is not recognized and the class re-assigned, an analysis with `lm()` will treat this as a regression instead of an ANOVA. The following example illustrates this.

An Example: Drainage Class

Let's illustrate how object classes are dealt with differently by looking at the drainage class of the stands in our Oak dataset. We begin by loading the data:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
```

Now we can explore the variable 'DrainageClass':

```
class(Oak$DrainageClass)
str(Oak$DrainageClass)
```

This variable has four unique values:

```
unique(Oak$DrainageClass)
```

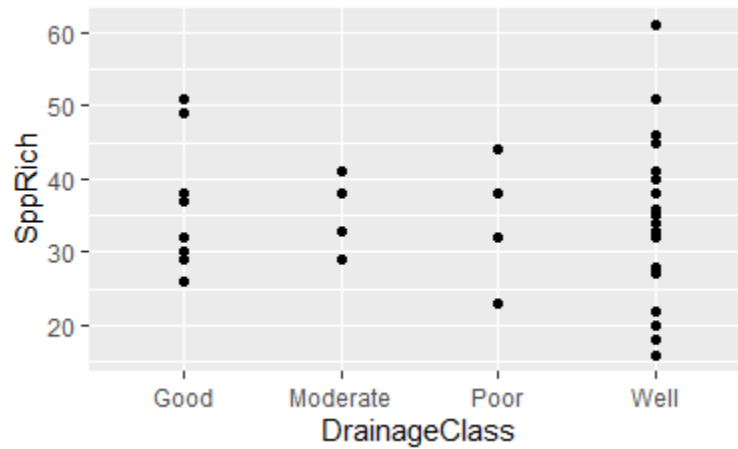
Is the difference between 'well' and 'poor' the same as that between 'well' and 'good'? No; these data are ordinal. To acknowledge this in R, we can change the class to recognize that these levels are ordered. For clarity, we'll create a new column that contains the same information but in a new class:

```
Oak$DrainageClass.Ordered <- factor(Oak$DrainageClass, ordered = TRUE, levels =
  c("Poor", "Moderate", "Good", "Well"))
```

Verify that the structure of the new column differs from that of the original column.

One way to see how R treats these objects differently is to graph them.

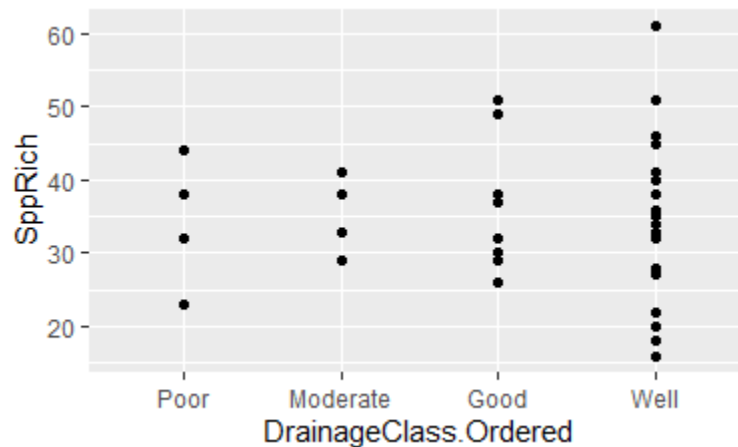
```
library(ggplot2)
ggplot(data = Oak, aes(x = DrainageClass, y = SppRich)) + geom_point()
```



Example graphic showing species richness in stands of differing drainage class, where those classes are organized in alphabetical order.

Does it make sense to arrange drainage classes in this way (i.e., alphabetically)? No.

```
ggplot(data = Oak, aes(x = DrainageClass.Ordered, y = SppRich)) + geom_point()
```



Example graphic showing species richness in stands of differing drainage class, where those classes are organized in order of increasing drainage.

Note that the levels on the x-axis are now in order of increasing drainage.

If we were confident that we did not need the data in the original class any more, we could have overwritten it by assigning the results of this command to that object. However, ordinal factors are analyzed via polynomials rather than via ANOVA:


```
summary(lm(SppRich ~ DrainageClass, data = Oak))
summary(lm(SppRich ~ DrainageClass.Ordered, data = Oak))
```

I therefore find it helpful to keep both variables so that I can use the unordered factor for analysis and the ordered factor for graphing.

Indexing

One powerful feature of R is the ability to **index** objects to extract or manipulate elements that meet certain criteria. The type of indexing depends on the class of the object.

With one-dimensional objects (which classes?), indexing is done by using square brackets to reference the desired element(s) by their position in the sequence. For example, if we want the fourth element in `y`:

```
y[4]
```

With two-dimensional objects (which classes?), indexing requires two pieces of information to reference the rows and columns of the desired element(s). This can be done using square brackets to refer to the [row, column] of interest. If we want all row or all columns, we can leave that part of the indexing blank – while still including the comma. Here, we will create a simple data frame containing `x`, `y`, and `z` – all of which were created above – and then index it.

```
xyz <- data.frame(cbind(x, y, z))
xyz[ , 2]          # all data from second column
xyz[ , "y"]        # ditto

xyz[2, ]           # all data from second row
xyz[c(2:3), ]      # data from second and third rows
xyz[-1, ]          # all data except the first row

xyz[2, 2]          # data from second row, second column
xyz[2, "y"]        # ditto
```

For clarity, it is recommended to include a space after the comma within the brackets. Some folk leave a space before the comma too. These spaces do not affect the indexing.

Columns within a data frame can also be indexed by name using the string symbol (`$`):

```
xyz$x
```

Note that this column is a one-dimensional vector, so if we are calling it then further indexing of elements within it are the same as for other one-dimensional objects:

```
xyz$x[3]
```

Key Takeaways

Index a one-dimensional object via `[item]`. For example, the 2nd observation in the object `x`: `x[2]`

Index a two-dimensional object:

- Via `[row, column]`. For example, the 2nd observation in the 1st row of column 1 in the object `xyz`:
`xyz[2, 1]`
- If the object has named columns, use `$` to return a column by name: `object$column_name`

When possible, index by name rather than by position.

Similarly, we can index our Oak data frame to focus on individual plots (rows) and/or variables (columns). Some examples:

```
Oak[, 1] # Refers to all data from column 1 (plot elevation)
```

```
Oak[, "Elev.m"] # Ditto
```

```
Oak$Elev.m # Ditto – but only works for data frame columns
```

```
Oak[1:2, ] # Refers to all data from rows 1 and 2
```

```
Oak[c("Stand01", "Stand02"), ] # Ditto
```

```
Oak[-5, 1] # All elevations except that in the 5th row
```

What is the elevation of stand 02? _____

Different types of indexing can be combined together. For example, we could choose particular columns to focus on and then index the rows of those columns. The next section presents three ways to conduct sequential actions on an object.

Conducting Multiple Sequential Actions

Often we want to conduct multiple actions in a sequence. For example, we might want to index a subset of the data that meets particular conditions. This can be done in at least three ways: as a series of separate actions, nested functions, and piping. I'll illustrate these using an example in which we want to calculate how many plots have an elevation above 100 m.

A Series of Separate Actions

The easiest way to solve this example is with a series of steps. Here's one solution:

```
oak_elevations <- Oak$Elev.m  
gt100 <- oak_elevations > 100  
sum(gt100)
```

```
[1] 35
```

The first step indexed the column of interest. The second step evaluated whether each value met the criterion – the resulting object, `gt100`, consists of a series of trues and falses. TRUE is automatically treated as 1 and FALSE as 0, so in the third step we summed them to produce the answer.

The disadvantage of this approach is that we've created intermediate objects (`oak_elevations`, `gt100`) for which we have no other use. Intermediate objects like this can be cumbersome if they are not also needed later in the code, and can make it difficult to keep track of the objects that we do want to use later. If there are small numbers of them, they can be manually removed:

```
rm(oak_elevations, gt100)
```

Nested Functions

By nesting one function inside another, we could have done all of these actions in a single step:

```
sum(Oak$Elev.m > 100)
```

```
[1] 35
```

Nested functions are common and very useful. However, they can be complicated to write, particularly because each function may have associated arguments that need to be referenced within the set of parentheses relating to that function – and it can be confusing to keep track of all those parentheses. For example, here is a more complicated example in which we index some data, relativize it, and then calculate the Euclidean distance between every pair of samples. We'll learn soon what relativizing means and what a Euclidean distance is, but for now just observe how I've written this code with indenting and colors to illustrate the nested functions and show the function to which each argument belongs:

```
library(vegan)
geog.dis <- vegdist(
  x = decostand(
    x = Oak[,c("LatAppx", "LongAppx")],
    method = "range"),
  method = "euc")
```

Here is the same code in a compact form (i.e., without indenting or named arguments):

```
geog.dis <- vegdist(decostand(Oak[,c("LatAppx", "LongAppx")], "range"), "euc")
```

Nested functions have several disadvantages:

- It can be confusing to keep track of which set of parentheses relates to which function.
- Interpreting code can be challenging: you have to start from the inner-most set of parentheses and work your way outward (in both directions!).
- Adjusting code can be challenging. In particular, if we want to remove an intermediate step in a series of nested functions, we need to determine which parentheses and arguments to remove. Imagine removing `decostand()` from the above code.

Piping

The **pipe** is an operator that allows you to feed the output of one function directly into another function. It is a technique that I find extremely helpful. See chapter 14 of Wickham & Grolemund (2017) for more information and examples of piping.

Piping is available through the pipe operator, `%>%`, in the `magrittr` package. Beginning in version 4.1.0, the 'forward pipe' (`|>`) is included in the base functionality of R. From what I have observed so far, these operators function identically.

The pipe is used at the end of one line to indicate that the output of that line is input for the next line. Specifically, the output of one line is input for the first argument of the function on the next line. Wickham & Grolemund (2017) describe it as focusing on verbs (actions), rather than nouns.

Applying similar actions to our data:

```
Oak |>
  filter(Elev.m > 100) |>
pull(Elev.m) |>
  length()
```

```
[1] 35
```

This may look more complicated than necessary for a simple example, but piping can be powerfully applied to more complicated sets of functions. For example, here is the more complicated example as in the nested functions above, and with the same color-coding:

```
library(magrittr)
geog.dis <- Oak[,c("LatAppx", "LongAppx")] |>
  decostand("range") |>
  vegdist("euc")
```

Note how much easier this is to read than a set of nested functions. The steps are listed in the order they are implemented, rather than having to read outward from the inner-most function. This makes the code easier to write and to tweak. For example, if I wanted to exclude one step from the set of functions, I can simply comment out that line.

One caveat is that not all functions work within pipes (this is why I didn't use the same functions to calculate the number of stands with elevations above 100 m), although I expect that to change now that piping is built into the base R installation – existing functions will be increasingly updated to work with the `|>` function.

Key Takeaways

Piping allows code to be easily read – steps are listed in the order they are implemented – and modified.

The `%>%` and `|>` operators are largely interchangeable.

When viewing code that includes a pipe, think of it as the phrase “and then”. For example, this code

```
takes the object Oak and then filters it to those observations with elevation above 100 m:  
Oak |> filter(Elev.m > 100)
```

Descriptive Statistics

Without indexing, functions are automatically applied to all columns in a matrix or data frame. This is evident if you use the `summary()` and `str()` functions:

```
summary(Oak)  
str(Oak)
```

Note that different types of data are summarized as appropriate, and that the class of each column is identified in its structure. Functions can be applied to rows within the `apply()` function.

Once we know how to index our data, we can apply functions to the selected elements:

```
summary(Oak[, 1]) # Returns range, mean, and quantiles  
summary(Oak[, "Elev.m"]) #equivalent - range, mean, and quantiles  
stem(Oak[, 1]) # Stem and leaf plot  
hist(Oak[, 1]) # Histogram
```

There are many other standard functions that can be used to examine your data: `max()`, `min()`, `quantile()`, `range()`, etc. You can also create your own functions – we'll explore this later.

Borcard et al. (2018) note the importance of exploratory data analysis – visualizations and simple statistics – when familiarizing yourself with a dataset. They illustrate how the above functions, and many others, can be used to do so. Zuur et al. (2010) also provide helpful ideas about exploratory data analysis.

Manipulating Data in R

The Tidyverse

Data can be manipulated using base R functions, but more intuitive means of doing so have been developed. In particular, Hadley Wickham and colleagues have produced the tidyverse, a number of 'opinionated' packages that share the same grammar. Key aspects include:

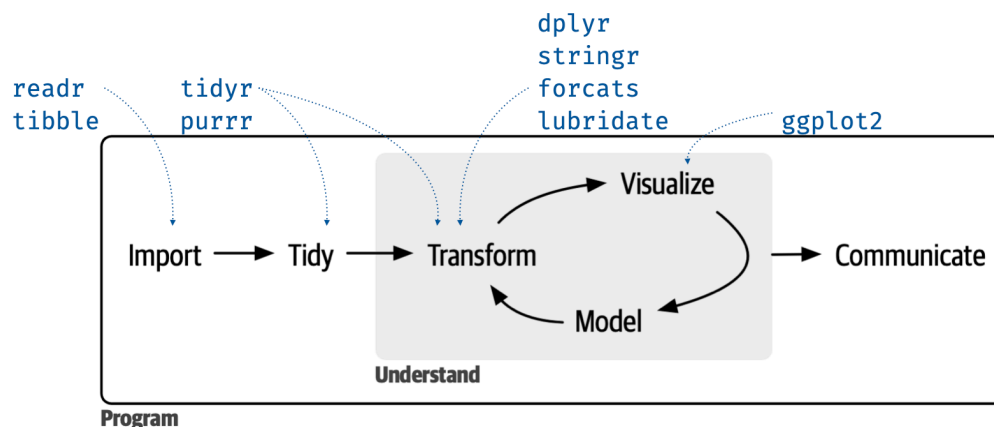
- Ability to pipe
- Don't need to put column names in quotation marks (usually) when calling them
- Consistent terminology that works across packages

These packages can be installed *en masse* by installing the `tidyverse` package. It includes the following core packages:

- `readr`, for data import
- `tibble`, for a modern re-imagining of data frames
- `dplyr`, for data manipulation
- `tidyr`, for data tidying
- `purrr`, for functional programming
- `ggplot2`, for data visualization
- `stringr`, for working with strings (characters)
- `forcats`, for working with factors
- `lubridate`, for working with dates and times



These packages are the focus of the book by Wickham & Grolemund (2017) and its associated website (or the second edition of this book, which has its own website). And, links to cheatsheets for most of these packages have also been provided above. One way to consider how these packages interrelate is as follows:



Linkages among tidyverse packages. From Çetinkaya-Rundel (2023).

Some additional packages are also available with more specialized applications – see the tidyverse website for details. In particular, I would highlight:

- `readxl`, for reading Microsoft Excel spreadsheets
- `googledrive`, for interacting with files stored on Google Drive
- `magrittr`, for the pipe (`%>%`). Note that this has been incorporated into the base functionality of new versions of R as `|>` and is being replaced by it.
- `tidymodels`, for modeling and machine learning. This is actually a collection of packages, like the core tidyverse. Most are for parametric models; I haven't investigated their application with the types of models that we focus on in this course.
- `broom`, for organizing model output

As an example of the tidyverse grammar, let's work with the stand-level attributes related to soil groups and soil series. Let's find out how many stands occur in each combination of soil group and soil series, and calculate summary statistics about species richness in each combination. We'll use piping so that the output of one function is the input of the next function.

```
Oak.soil.sum <- Oak %>%
  group_by(SoilGroupName, SoilSeriesName) %>%
  summarize(N = length(SppRich),
    MinS = min(SppRich),
    MeanS = round(mean(SppRich), 1),
    MaxS = max(SppRich),
    .groups = "keep")
Oak.soil.sum
```

```
# A tibble: 7 × 6
# Groups:   SoilGroupName, SoilSeriesName [7]
  SoilGroupName SoilSeriesName      N  MinS MeanS  MaxS
  <chr>         <chr>         <int> <int> <dbl> <int>
1 Alluvial      Amity             3     32    38     44
2 Basic.igneous Dixonville      5     27   38.4    61
3 Basic.igneous Nekia           4     32    39     51
4 Basic.igneous Olympia         2     28    31     34
5 Sedimentary  Carlton          6     23   29.2    35
6 Sedimentary  Peavine            5     33   37.4    41
7 Sedimentary  Steiwer           22     16   33.3    51
```

Note that we created, and named, new variables for the number of stands (N) in each group along with the minimum, average, and maximum species richness in each group.

This type of summary is helpful both when manipulating data and when summarizing it for graphing purposes, etc.

plyr

The `plyr` package is related to the tidyverse (but not included in it), and is one I've found particularly helpful. The basic process within `plyr` is to split an object, do something to it, and then re-combine it (Wickham 2011).

There are several functions within `plyr` that have the same structure and naming convention:

- The first letter (a, d, or l) indicates whether it starts with an array, data frame, or list
- The second letter (also a, d, or l) indicates whether it produces an array, data frame, or list.

For example, `ddply()` takes a data frame, does something to it, and produces a new data frame. Its usage is:

```
ddply(.data, .variables, .fun = NULL, ..., .progress = "none", .inform = FALSE,
.drop = TRUE, .parallel = FALSE, .paropts = NULL)
```

The key arguments are:

- `.data` – the data frame to be processed
- `.variables` – the variables to use when splitting the data frame
- `.fun` – the function to apply to each piece after splitting

New variables can be created and named as part of this process.

This package uses the period function (`.`) to allow column names to be specified without having to place them in quotation marks.

As an example, let's do the same calculation of species richness in each combination of soil group and soil series:

```
library(plyr)
Oak.soil.sum <- ddply(.data = Oak,
  .variables = .(SoilGroupName, SoilSeriesName),
  .fun = summarize,
    N = length(SppRich),
    MinS = min(SppRich),
    MeanS = round(mean(SppRich), 1),
    MaxS = max(SppRich))
Oak.soil.sum
```

	SoilGroupName	SoilSeriesName	N	MinS	MeanS	MaxS
1	Alluvial	Amity	3	32	38.0	44
2	Basic.igneous	Dixonville	5	27	38.4	61
3	Basic.igneous	Nekia	4	32	39.0	51
4	Basic.igneous	Olympia	2	28	31.0	34
5	Sedimentary	Carlton	6	23	29.2	35
6	Sedimentary	Peavine	5	33	37.4	41
7	Sedimentary	Steiwer	22	16	33.3	51

This object is identical to that produced via piping, except that it is saved as an object of class 'data.frame' instead of class 'tibble'.

Of course, this like all other code can be streamlined by omitting the argument names if the arguments are called in order:

```
Oak.soil.sum <- ddply(Oak, .(SoilGroupName, SoilSeriesName), summarize,
  N = length(SppRich),
```



```

MinS = min(SppRich),
MeanS = round(mean(SppRich), 1),
MaxS = max(SppRich))
Oak.soil.sum

```

merge()

An essential function to be familiar with when you want to combine objects is `merge()`, which allows you to combine two objects on the basis of one or more specified columns. This is a base R function. Its usage is:

```

merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, all =
  FALSE, all.x = all, all.y = all, sort = TRUE, suffixes = c(".x", ".y"), no.dups =
  TRUE, incomparables = NULL, ...)

```

The key arguments are:

- `x, y` – the data frames to be combined. Note that if you want to only include certain columns from a data frame you can index them here. Be sure, however, that you index it to include both the column(s) that form the basis for merging (needed for the `by` argument) and the column(s) that you want to include in the resulting object.
- `by, by.x, by.y` – the columns to use as the basis for merging. If the columns have the same name, you can use the `by` argument. If the column names differ between objects, specify the relevant columns via `by.x` and `by.y`. Merging can also be done on the basis of row names by setting any of these equal to `"row.names"`.
- `all, all.x, all.y` – whether to include rows that do not have a match in the other object. As for the `by` argument, the `all` argument keeps all rows from both objects while `all.x` only keeps those from `x` and `all.y` only keeps those from `y`. The default (`all = FALSE`) is to keep only those rows that are present in both objects.
- `suffixes` – text to append to the end of column names when column(s) with the same name occur in both data frames but are not part of the criteria for merging. Sometimes it is helpful to change the defaults so that column names from one data frame are unchanged after merging and thus still work with your previous code. For example, to keep column names from `x` unchanged while distinguishing those from `y`, we could set `suffixes = c("", ".y")`.

As an example, let's associate the mean species richness values for each soil group and soil series (calculated above) with the individual observations:

```

Oak$Stand <- row.names(Oak)
Oak.new <- merge(
  x = Oak[, c("SoilGroupName", "SoilSeriesName", "Stand", "SppRich") ],
  by = c("SoilGroupName", "SoilSeriesName"),
  all = TRUE,
  y = Oak.soil.sum[, c("SoilGroupName", "SoilSeriesName", "MeanS") ] )

```

Some notes about this code:

- I indexed `x` and `y` to only include the variables being used for merging and the variables that I wanted to keep in the new object (`Oak.new`). If I did not do this, then all of the columns from both objects would have been retained.
- I was able to use the `by` argument (rather than `by.x` and `by.y`) because the columns forming the basis of the merge had the same name in both objects.
- When these objects do not contain the exact same set of observations, the `all` argument and its variations are important. In this case, every observation was present in both objects.

Viewing the first few rows of output:

```
head(Oak.new)
```

	SoilGroupName	SoilSeriesName	Stand	SppRich	MeanS
1	Alluvial	Amity	Stand01	32	38.0
2	Alluvial	Amity	Stand46	38	38.0
3	Alluvial	Amity	Stand44	44	38.0
4	Basic.igneous	Dixonville	Stand42	33	38.4
5	Basic.igneous	Dixonville	Stand41	38	38.4
6	Basic.igneous	Dixonville	Stand12	33	38.4

One very helpful way to diagnose functions like this is to keep track of the dimensionality of the starting and ending objects. Here, we began with a 47 x 4 object (four columns from `Oak`) and a 7 x 3 object (3 columns from `Oak.soil.sum`), and ended with a 47 x 5 object – the number of columns increased by one because we used two columns to index the two objects and added a single column of data. The number of rows didn't change because every observation from `Oak` matched an observation from `Oak.soil.sum`.

Merging can also be applied within piping – the value that is fed into `merge()` is assumed to belong to the first argument (`x`). I'll illustrate this by doing the same calculations as above while also using some additional tidyverse functions:

```
Oak.new <- Oak %>%
  mutate(Stand = row.names(Oak)) %>%
  dplyr::select(SoilGroupName, SoilSeriesName, Stand, SppRich) %>%
merge(by = c("SoilGroupName", "SoilSeriesName"),
all = TRUE,
y = Oak.soil.sum[, c("SoilGroupName", "SoilSeriesName", "MeanS") ])
```

Verify that the resulting object is the same.

The `dplyr` package also includes a family of functions to join (merge) objects:

- `inner_join(x, y)`: only include observations that match in both `x` and `y`.
- `left_join(x, y)`: include all observations of `x` regardless of whether they match `y`. `x` is assumed to be your primary table, so this is the most commonly used join. If a row doesn't match, the new variables are filled in with missing values.
- `right_join(x, y)`: include all observations of `y` regardless of whether they match `x`. If a row doesn't match, the new variables are filled in with missing values.
- `full_join(x, y)`: include all observations of `x` and of `y`. If a row doesn't match, the new variables are filled in with missing values.

More information on these and other “two-table verbs” is available at <https://dplyr.tidyverse.org/articles/two-table.html>.

Conclusions

Understanding how to name and manipulate objects in R is foundational for all other aspects of this course. Much of the debugging that inevitably occurs during coding relates to how data are being manipulated.

References

- Adler, J. 2010. *R in a nutshell: a desktop quick reference*. O'Reilly, Sebastopol, CA.
- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Çetinkaya-Rundel, M. 2023. Teaching the tidyverse in 2023. <https://www.tidyverse.org/blog/2023/08/teach-tidyverse-23/>
- Crawley, M.J. 2012. *The R book*. Second edition (First edition, 2007). Wiley, West Sussex, England.
- Dalgaard, P. 2008. *Introductory statistics with R*. Second edition (First edition, 2002). Springer, New York, NY.
- Filazzola, A., and C.J. Lortie. 2022. A call for clean code to effectively communicate science. *Methods in Ecology and Evolution* 13:2119-2128.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- Paradis, E. 2005. *R for beginners*. Université Montpellier II, Montpellier, France. http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- Robinson, A. 2016. *icebreakR*. <https://cran.rstudio.com/doc/contrib/Robinson-icebreaker.pdf>
- Sellers, M. 2019. *Field guide to the R ecosystem*. <http://fg2re.sellorm.com/>
- Torfs, P., and C. Brauer. 2014. *A (very) short introduction to R*. <http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf>
- Venables, W.N, D.M. Smith, and the R Development Core Team. 2023. *An introduction to R*. Version 4.3.2. <http://cran.r-project.org/doc/manuals/R-intro.pdf>
- Wickham, H. 2011. The split-apply-combine strategy for data analysis. *Journal of Statistical Software* 40:1-29. <http://www.jstatsoft.org/v40/i01/>.
- Wickham, H. 2014. *Advanced R*. Chapman & Hall/CRC, Boca Raton, FL.
- Wickham, H. 2020. *The tidyverse style guide*. <https://style.tidyverse.org/>
- Wickham, H., and G. Grolemund. 2017. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly, Sebastopol, CA. <http://r4ds.had.co.nz/>
- Zuur, A.F., E.N. Ieno, and C.S. Elphick. 2010. A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution* 1:3-14.

Media Attributions

- drainage.unordered
- drainage.ordered
- tidyverse
- tidyverse_packages © Çetinkaya-Rundel

5. Data Adjustments

Learning Objectives

- To present ways to identify data entry errors.
- To consider when and how to script temporary data adjustments.
- To begin considering how data adjustments relate to the research questions being addressed.
- To continue using R, including functions to adjust data.

Readings

Broman & Woo (2018)

Key Packages

```
require(tidyverse, labdsv)
```

Properties of Multivariate Ecological Data

Many ecological studies collect data about multiple variables on the same sample units. Manly & Navarro Alberto (2017, ch. 1) provide several examples of multivariate datasets. Types of data that have been the foci of analyses in this course include:

- Physiology: gas exchange, photosynthesis, respiration, stomatal conductance
- Plant ecology: abundances of multiple species
- Fire ecology: tree mortality, scorch height, fuel consumption, maximum temperature
- Remote sensing: height of LiDAR returns, canopy density, rumple
- Anatomy: morphometric measurements
- Bioinformatics / genomics: genetic sequence data

Since these data are collected on the same units, they are multivariate. However, they often have many properties that confound conventional multivariate analyses:

- **Bulky** – large number of elements in a matrix
- **Intercorrelated / redundant** – for example, species often respond to the same gradients, and LiDAR-derived metrics may be highly correlated with one another
- **Outliers** – non-normal distributions. For example, species composition data is notorious for having lots of zeroes (i.e., species absent from plots)

Given these properties, it is often necessary to adjust data prior to analysis. McCune (2011) provides a decision tree that summarizes multiple options for data adjustment and for analyses. Other suggestions are provided in an appendix to these notes.

Data Organization (redux)

As discussed in the context of reproducible research, scripting is a powerful ability that distinguishes R from point-and-click statistical software. When a script is clearly written, it is easy to re-run an analysis as desired – for example, after a data entry error is fixed or on a particular subset of the data.

Once your data have been entered and corrected, all subsequent steps should be automated in a script. This often requires data manipulation, such as combining data from various subplots, or calculating averages across levels of a factor. New R users often want to export their data to Excel, manipulate it there, and re-import it into R. **Resist this urge.** Working in R helps you avoid errors that can creep in in Excel – such as calculating an average over the wrong range of cells – and that are extremely difficult to detect. If tempted to export your data file to Excel so that you can re-organize or summarize it, I encourage you to first check out the R help files (including on-line resources), ask colleagues how they solve similar issues, and consult resources like Wickham & Grolemund (2017). Packages like `dplyr` and `tidyr` are particularly helpful for these types of tasks.

The general principle is to **maintain a single master data file**. Obvious errors should be permanently corrected in this master data file, but other changes may be context dependent. Scripting these context-dependent changes enables you to:

- work from the same raw data every time
- avoid having multiple versions of your data file

Broman & Woo (2018) provide many valuable suggestions for organizing data within spreadsheets.

By convention, multivariate data are organized in a matrix where each row is a unique sample unit and each column is a unique variable. This is consistent with the ‘tidy data’ philosophy within the tidyverse:

- Each variable is its own column. I often refer to the columns as ‘species’ for simplicity, but they might be measured values or assigned values such as plot ID, year, experimental treatment etc.
- Each sample unit is its own row. I often refer to the rows as ‘plots’ for simplicity, but they might be stands or plot-year combinations (i.e., a measurement of a given plot at a point in time) or other units of observation. The units will depend on the study system, but it is essential that each row be uniquely identified.

This structure for organizing data is called the ‘wide’ format in `tidyr`. In practice, data are sometimes stored in a ‘long’ or **triplet form** where one column identifies the sample unit, one column identifies the variable, and one column identifies the value of that variable in that sample unit. These organizational structures are related, as illustrated in the below figure.

country	1999	2000	→	country	year	cases
A	0.7K	2K		A	1999	0.7K
B	37K	80K		B	1999	37K
C	212K	213K		C	1999	212K
				A	2000	2K
				B	2000	80K
				C	2000	213K

Comparison of the 'wide' and 'long' (or triplet) structures for organizing data. In this case, the countries are sample units and the years (1999, 2000) are the variables. From Posit (2023).

The `tidyr` package includes the `pivot_wider()` function to convert data from long to wide format, and the `pivot_longer()` function to convert from wide to long format.

Triplet form can be a particularly efficient way to store compositional data as many species are often absent from most sample units. If all non-zero values are included in the triplet form, the zeroes can be inferred and automatically added when data are converted to wide format.

Data Organization

Store data in master data files, with sample units as rows and variables as columns. Each row should have a unique identifier, and each column should contain a single variable.

Describe the rows and columns of the master data file in a metadata document.

Ideally, prepare a single master data file for data collected at each spatial scale. Apply unique identifiers consistently across data files so they can be merged within R.

Identify and Permanently Correct Data Entry Errors

I am assuming here that the data we are using are organized in one or more spreadsheets; Broman & Woo (2018) provide valuable ideas on how to do so.

It may seem self-evident, but it's important to check for and correct data errors before proceeding to analysis. Data cleaning can be more time-consuming than the actual analyses that follow (de Jonge & van der Loo 2013). Although I rely exclusively on R once I'm ready for analysis, I generally do this initial type of quality control in Excel (via Pivot Tables, etc.) before importing data into R. However, de Jonge & van der Loo (2013) provide detailed notes about how to explore and clean up data within R. Zuur et al. (2010) provide very helpful suggestions for data exploration.

Ways to identify data entry errors include:

- Recheck the data entry on a randomly selected subset of the data. This can be done by yourself or by having someone read the data back to you.
- Decide how to deal with missing data. See suggestions of McCune & Grace (2002, p. 58-59) and de Jonge & van der Loo (2014, ch. 3).
- Examine the minimum and maximum values to verify that they span the permitted range of values.
- If dealing with a categorical variable, view the unique values recorded and make sure they are all appropriate. Watch out for seemingly minor differences:
 - A plot with a topographic position of 'slope' will be assigned to a different category than one with a position of 'Slope' (note the capital!).
 - Similarly, 'slope' and 'slope ' would be treated as two different values because of the space in the latter.
- Plot reasonably correlated variables against one another to identify potential bivariate outliers. These might reflect data entry errors, though they may also be true outliers (see below). Anomalies can also sometimes be detected by calculating the ratio of two variables (e.g., tree height and diameter) and then looking at the extreme values in the distribution of this ratio. This technique is less appropriate for community data than for other types of data (e.g., environmental variables that are correlated with one another).
- If you are dealing with compositional data, resolve unknowns and assign each unknown species to a unique species code. Be sure to keep track of what you've done – you may encounter that unknown code again elsewhere! It is often helpful to save two species codes for each entry: one for the original field code, and one for the code to be analyzed.

In spite of our best efforts at quality control, errors sometimes slip through. This is where scripts really shine. If you have kept careful track of the steps in your analysis (see the 'Reproducible Research' chapter), once you have corrected these errors you will be able to easily re-do your analyses, re-create graphics, etc. simply by re-running the script.

Related to this is the issue of detecting outliers. Univariate outliers are relatively easy to detect. However, we have already noted that community ecology data are often non-normally distributed – in particular, there is usually a plethora of zeros (absences) in a site × species matrix. These might appear to be outliers statistically, but they aren't outliers biologically and shouldn't be treated as such!

Multivariate outliers are much more difficult to detect. One way to do so is to compare the range of dissimilarities among elements. See here for more detail.

An interesting non-R option for cleaning up messy data is OpenRefine (<https://openrefine.org/>). I have not used this website, however, so cannot speak to its effectiveness.

Scripted (Temporary) Data Adjustments

Often we may want to adjust the data for an analysis but do not want to permanently change the original data file. We will consider three types of temporary adjustments here: consolidating taxonomy, deleting sample units, and deleting rare species. These types of adjustments are often required for compositional analyses but may be less common for other types of multivariate data.

Scripted Adjustments to Data

Scripts provide a means of adjusting the data for an analysis without permanently changing the original data file. Examples of these adjustments include:

- Consolidating taxonomy
- Deleting sample units (rows) – either because they are empty or to focus on a subset of them
- Deleting species (columns) because they are absent from the focal sample units or are rare and assumed to contribute little to the patterns among sample units

Your analytical methods should clearly identify whether and which adjustments were made.

Consolidating Taxonomy

It can be difficult to identify the taxa present in an ecological study, and people vary in their taxonomic proficiency. In one study (Morrison et al. 2020), more than half of the apparent differences in cover of plant species over time were identified as observer errors! As another example, I analyzed compositional variation in grasslands around the world (Bakker et al. 2023). Each site has multiple plots, and each plot was assessed in multiple years. As part of my quality control, I reviewed which species were identified within each site and year and highlighted inconsistencies that likely reflect the person doing the work (as in the example below) rather than actual ecological changes. My analysis includes scripted adjustments to taxonomy at 70% of the 60 sites in the analysis, indicating that this is not a rare scenario.

Consider the following example of species composition in two plots in two years:

Year	Plot	<i>Poa compressa</i>	<i>Poa pratensis</i>	<i>Poa</i> spp.
1	i	1	3	
1	ii	2	2	
2	i			4
2	ii			4

This might happen, for example, if the person who assessed composition in year 2 was less confident in their ability to distinguish the two *Poa* spp. than the person who assessed composition in year 1.

Analyses that focus just on Year 1 would likely want to include the fact that two species of *Poa* were documented that year. For example, these species might respond differently to environmental conditions or to experimental factors.

What about comparisons between Years 1 and 2? As reported here, each plot has lost two species (*Poa compressa* and *P. pratensis* were present in Year 1 but absent in Year 2) and gained one species (*Poa* spp.). However, it should be clear that these are differences in taxonomic resolution rather than meaningful ecological changes. Instead, it would be reasonable to combine *P. compressa* and *P. pratensis* for analyses that compare the years. This can be easily scripted:

- Add the covers of *P. compressa*, *P. pratensis*, and *Poa* spp., and assign the summed cover to a column with a new taxonomic code, such as 'Poa_combined'.
- Delete the columns for *P. compressa*, *P. pratensis*, and *Poa* spp.

To run this in R, we'll first create our dataset and then use the `mutate()` function from the `tidyverse`. Of course, there are many other ways that this could also be scripted.

```
poa_example <- data.frame(
  Year = c(1, 1, 2, 2),
  Plot = c("i", "i", "ii", "ii"),
  Poa_compressa = c(1, 2, 0, 0),
  Poa_pratensis = c(3, 2, 0, 0),
  Poa_spp = c(0, 0, 4, 4))

library(tidyverse)

poa_adjusted <- poa_example %>%
  mutate(Poa_combined = Poa_pratensis + Poa_compressa + Poa_spp,
    .keep = "unused")
```

	Year	Plot	Poa_combined
1	1	i	4
2	1	i	4
3	2	ii	4
4	2	ii	4

Note that the abundance of 'Poa_combined' is identical in all plots and years of this simple example; this was not obvious from the original data.

Deleting Sample Units (Rows)

With some types of ecological data, it is possible for sample units to be empty – for example, there might be no species growing in a vegetation plot. While this information is ecologically relevant, many analytical techniques require that every row of the matrix contain at least one non-zero value. There are two common ways to deal with empty sample units:

- Delete empty sample units from the analysis. This is not ideal for several reasons:
 - It reduces the size of the dataset and can alter the balance of our design.
 - It changes the questions being asked. For example, imagine that we assessed plots in two habitats, and that many of the sample units in one of the habitats were empty. Using all of the sample units, we can ask whether composition differs between habitats. Using only those sample units that contain species, we can ask whether composition differs between habitats given that we are only considering areas that contain species.
- Add a dummy variable with a small value to all sample units (Clarke et al. 2006). Doing so means that all sample units have at least one non-zero variable, permitting their inclusion in analyses that require data in all rows. I'll illustrate this approach in the chapter about distance measures.

If data are stored in multiple objects, such as when response variables are in one object and potential explanatory variables are in another, we need to apply changes to both objects. If we delete a sample unit (row) from one object, we also need to remove that sample unit from the other object. And, of course we need to make sure that the remaining sample units are in the same order in both objects.

Another reason to delete sample units is to focus on a subset of the data for a particular analysis.

The `filter()` function from the `tidyverse` is very useful here. For example, if we wanted to focus on Year 1 from `poa_example` above:

```
poa_example %>% filter(Year == 1)
```

	Year	Plot	Poa_pratensis	Poa_compressa	Poa_spp
1	1	i	1	3	0
2	1	i	2	2	0

Deleting Species (Columns)

Sometimes a dataset includes columns that are empty. For example, this often happens with compositional data if you've filtered the data to focus on a subset of the sample units; some species were never recorded in the subset that you're focusing on. These 'empty' species are less of a computational concern than empty sample units, though relativizations that are applied to columns (see 'Relativizations' chapter) can produce 'NaN' values if applied to empty columns. It is also computationally inefficient to include a large number of empty columns, though computing power is no longer a significant consideration for most analyses. We can delete these empty columns before proceeding with analyses.

A more intriguing instance can arise with compositional data: **some people suggest that we delete rare species**. It may seem counter-intuitive to ignore these species, but the logic is that rare species are often poorly sampled and therefore contribute little information about the relationship with environmental gradients, etc. (I suspect that concerns about computing power were also a factor in developing this recommendation, but are no longer relevant).

When is a species rare? A cutoff of **5% of sample units** is often adopted. So, if a dataset contains 20 sample units, the 5% rule indicates that species that only occur in a single sample unit should be deleted. If a dataset contains 40 sample units, species that occur in one or two sample units would be deleted.

When data have been collected in distinct areas (e.g., experimental treatments), deletion could occur on the basis of the full dataset (i.e., species that occur on < 5% of all sample units) or on individual treatments (i.e., species that occur on < 5% of the sample units in a given treatment). Generally, however, sample sizes are too low for the latter approach to be meaningful.

Surprisingly few studies have examined the effect of removing rare species. Cao et al. (2001) noted that consideration of the dominant species often provides insight into major environmental gradients, and concluded that "the theoretical or empirical justification for deleting rare species in bioassessment is weak". Poos & Jackson (2012) used electrofishing to sample the fish community at each of 75 sites in a watershed. They analyzed these data after making four types of adjustments: i) using the full dataset (no species removed), ii) removing species occurring at a single site, iii) removing those occurring at less than 5% of sites, and iv) removing those occurring at less than 10% of sites. The effect of these removals was compared with the effects of choice of distance measure and choice of ordination method, both of which we'll discuss later. They concluded that removal of rare species had about as large an effect on the conclusions as did the choice of ordination method. Brasil et al. (2020) examined how the compositional variation within fish and insect communities was affected by systematically removing rare or common species (though they defined rarity by abundance rather than by the number of sample units in which a species was present). They concluded that the proportion of compositional variation that could not be explained by their explanatory variables – aka the residual variation – was minimally affected even by the removal of half of the species. This was especially true if variation was expressed on the basis of abundance rather than presence. Further examination of this topic is clearly warranted (anyone want to include

this in their project?). As a counter to this focus on rare species, Avolio et al. (2019) provide a nice consideration of the importance and identification of dominant species.

There is no requirement that rare species be removed from a community ecology study; it is up to you to decide whether to do so or not. Of course, you should clearly state whether you did, why, and what your minimum rarity threshold was.

Please note that the above considerations are not applicable in all circumstances. Rarity in this context refers to sample units where a species was not detected. If your response variables are not species, then your data may contain values for all sample units. And, even if there are some rare 'species', it may not make sense to remove them. For example, if your response variables were measures of burn severity, it likely would make sense to retain all burn severities even if they were not detected in your particular study.

Finally, bioinformatics studies include pipelines that include pre-processing steps which can effectively remove species from a dataset, particularly for speciose systems such as microbial systems (Zhan et al. 2014).

Application in R

Rare species (response variables; columns) can be removed manually, but it is easier to use an existing function. We'll illustrate this using our sample dataset.

After opening your R project, load the sample dataset:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
```

Create separate objects for the response and explanatory data:

```
Oak_abund <- Oak[, colnames(Oak) %in% Oak_species$SpeciesCode]
Oak_explan <- Oak[, ! colnames(Oak) %in% Oak_species$SpeciesCode]
```

See the 'Loading Data' chapter if you do not understand what these actions accomplished.

`Oak_abund` contains species abundance data for 189 species encountered on the 47 stands. Therefore, a 5% cutoff ($47 \times 0.05 = 2.35$) means deleting species that occur in less than 3 stands.

labdsv::vegtab()

This function eliminates rare species and re-orders the columns and/or rows in a data frame. The help file for this function shows its usage:

```
vegtab(comm, set, minval = 1, pltord, spcord, pltlbl, trans = FALSE)
```

The key arguments are:

- `comm` – the data frame containing the data to be analyzed
- `minval` – the minimum number of observations that a variable has to occur to be retained. The default is 1 observation (i.e., empty variables – those without any occurrences – are deleted)
- `pltord` – a numeric vector specifying the order of rows in the output
- `spcord` – a numeric vector specifying the order of columns in the output

To use this function, we need to specify the dataframe (`comm` argument) and the minimum number

of observations (`minval` argument). Since we are not seeking to re-order our dataframe, we do not need to specify the `pltord` or `spcord` arguments.

```
Oak_no_raw <- vegtab(comm = Oak_abund, minval = (0.05 * nrow(Oak_abund)))
```

Note how I set `minval` equal to 5% of the number of samples, without specifying how many samples there were. This approach provides flexibility as it remains correct even if the number of rows in the dataset changes. If I didn't need this flexibility, I could equivalently have hard-coded the minimum number of observations (`minval = 3`).

How many species are in the reduced dataset? _____

How many species were in the original dataset? _____

How many species were removed? _____

For clarity, I named each argument explicitly in the above code. If we list the arguments in the order specified by a function's usage, we can omit the argument names. Or, we can just use a smaller unique portion of the argument name and let R match it against the arguments to figure out which one we mean:

```
Oak_no_raw <- vegtab(Oak_abund, min = (0.05 * nrow(Oak_abund)))
```

While these usages are more compact, they can be frustrating if you have to figure out later which arguments are being used. I specify arguments by name to avoid this.

Once you've removed rare species, you may have to check again for empty sample units.

Concluding Thoughts

Decisions about how to adjust data – including whether to consolidate taxonomy, focus on a subset of the sample units, and to delete rare species – can strongly affect the conclusions of subsequent analyses.

Empty sample units and rare species should never be deleted when analyzing species richness or diversity. In these calculations, the rare species are of critical importance, and a sample unit containing no species will affect the mean species richness, the slope of a species-area curve, etc. This is another reason to script these adjustments and preserve the original data.

References

Avolio, M.L., E.J. Forrester, C.C. Chang, K.J. La Pierre, K.T. Burghardt, and M.D. Smith. 2019. Demystifying dominant species. *New Phytologist* 223:1106-1126.

Bakker, J.D., J.N. Price, J.A. Henning, E.E. Batzer, T.J. Ohlert, C.E. Wainwright, P.B. Adler, J. Alberti, C.A. Arnillas, L.A. Biederman, E.T. Borer, L.A. Brudvig, Y.M. Buckley, M.N. Bugalho, M.W. Cadotte, M.C. Caldeira, J.A. Catford, Q. Chen, M.J. Crawley, P. Daleo, C.R. Dickman, I. Donohue, M.E. DuPre, A. Ebeling, N. Eisenhauer, P.A. Fay, D.S. Gruner, S. Haider, Y. Hautier, A. Jentsch, K. Kirkman, J.M.H. Knops, L.S. Lannes, A.S. MacDougall, R.L. McCulley, R.M. Mitchell, J.L. Moore, J.W. Morgan, B. Mortensen, H. Olde Venterink, P.L. Peri, S.A. Power, S.M. Prober, C. Roscher, M. Sankaran, E.W. Seabloom, M.D. Smith, C. Stevens, L.L. Sullivan, M. Tedder, G.F. Veen, R. Virtanen, and G.M. Wardle. 2023. Compositional variation in grassland plant communities. *Ecosphere* 14(6):e4542. <https://doi.org/10.1002/ecs2.4542>.

Brasil, L.S., T.B. Vieira, A.F.A. Andrade, R.C. Bastos, L.F.d.A. Montag, and L. Juen. 2020. The importance of common and the irrelevance of rare species for partition the variation of community matrix:

implications for sampling and conservation. *Scientific Reports* 10:1977. Doi: 10.1038/s41598-020-76833-5

Broman, K.W., and K.H. Woo. 2018. Data organization in spreadsheets. *The American Statistician* 72:2-10. Doi:10.1080/00031305.2017.1375989

Cao, Y., D.P. Larsen, and R.St-J. Thorne. 2001. Rare species in multivariate analysis for bioassessment: some considerations. *Journal of the North American Benthological Society* 20:144-153.

Clarke, K.R., P.J. Somerfield, and M.G. Chapman. 2006. On resemblance measures for ecological studies, including taxonomic dissimilarities and a zero-adjusted Bray-Curtis coefficient for denuded assemblages. *Journal of Experimental Marine Biology and Ecology* 330:55-80.

de Jonge, E., and M. van der Loo. 2013. *An introduction to data cleaning with R*. Statistics Netherlands, The Hague, Netherlands. 53 p. http://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf

Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.

McCune, B. 2011. *A decision tree for community analysis* [poster]. MjM Software Design, Gleneden Beach, OR.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Morrison, L.W., S.A. Leis, and M.D. DeBacker. 2020. Interobserver error in grassland vegetation surveys: sources and implications. *Journal of Plant Ecology* 13(5):641-648.

Poos, M.S., and D.A. Jackson. 2012. Addressing the removal of rare species in multivariate bioassessments: the impact of methodological choices. *Ecological Indicators* 18:82-90.

Posit. 2023. Data tidying with `tidyr` :: cheatsheet. <https://rstudio.github.io/cheatsheets/tidyr.pdf>

Wickham, H., and G. Grolemund. 2017. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly Media, Sebastopol, CA. <http://r4ds.had.co.nz/>

Zhan, A., W. Xiong, S. He, and H.J. MacIsaac. 2014. Influence of artifact removal on rare species recovery in natural complex communities using high-throughput sequencing. *PLoS ONE* 9(5):e96928.

Zuur, A.F., E.N. Ieno, and C.S. Elphick. 2010. A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution* 1:3-14.

Media Attributions

- `long.v.wide.table.format`

6. Transformations

Learning Objectives

To consider how transformations relate to the research questions being addressed.

To illustrate how to transform data in R.

Monotonic Transformations

Monotonic transformations are **applied identically** to all data elements. This means that the action taken on an individual element is unchanged whether you consider it alone or as part of a set. For example, calculating the square root of a value is unaffected by whether that value is part of a set. This is contrasted with relativizations, where the result of the action depends on other elements in the set.

There are many potential transformations that can be applied to data; we will review the most common ones here. McCune & Grace (2002, p. 67) note that transformations can be conducted for statistical or ecological reasons. However, many of the techniques we will cover do not require normality and other assumptions of parametric techniques. Thus, we can focus our transformations on the ecological questions that we seek to answer.

If you apply a transformation to univariate data such as an explanatory variable, the data should generally be back-transformed to the original units for presentation – as is true for all types of analyses.

Roots (square root, cube root, etc.)

Root transformations can be applied to count data, which generally follow a Poisson distribution. Vegetation work rarely uses higher-order roots, but studies in other systems do. For example, marine benthic studies may include organisms from phyla that span several orders of magnitude in abundance – there might be one starfish but tens of thousands of smaller invertebrates. Fourth-root transformations are often applied to this type of data so that the numerically dominant smaller taxa do not overwhelm comparisons among sample units.

Logarithms

Biomass or ratio data are often log-transformed. This commonly involves base-10 or natural logarithms (make sure to note which you use!).

If your data include zeroes, you may need to add a small value to all data because you can't calculate $\log(0)$. This can be done manually, or you can use an existing function such as `log1p()`.

Arcsin-square root

The arcsin-square root transformation is used with proportional data such as percent cover. It doesn't work for negative values or values > 1 .

Some authors strongly discourage using this transformation for univariate analyses (Warton & Hui 2011), but I have not seen this recommendation carried over to multivariate contexts.

Binary

A binary transformation converts continuous data to 0 or 1 based on whether a criterion is met. This is often used to convert abundance data to presence/absence. Another example of this as a transformation would be to evaluate whether abundance data exceed a static value such as '5 individuals' or '5% cover'.

Depending on the criterion, a binary adjustment can also be a type of relativization (see the 'Relativizations' chapter).

Transformations

Transformations are applied identically to all elements within an object.

Applications in R

In R, transformations are easily performed by applying a function to a matrix; the function is automatically applied to every element in the matrix. The transformed data are generally assigned to a new object so that the original data remain intact.

Here are the above transformations:

R Function	Note
<code>sqrt(x)</code> or <code>x^(1/2)</code>	Square root of x
<code>x^(1/4)</code>	Fourth root of x
<code>log10(x)</code>	Logarithm (base 10) of x
<code>log(x)</code>	Natural logarithm of x
<code>log1p(x)</code>	Natural logarithm of $x + 1$
<code>asin(sqrt(x))</code>	Arcsin square root of x
<code>ifelse(x > 0, 1, 0)</code>	Convert x to presence/absence

Oak Plant Communities Example

Let's illustrate these transformations using our oak plant communities dataset. Begin by opening the R project and the loading the data:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
```

Create separate objects for the response and explanatory data:

```
Oak_abund <- Oak[, colnames(Oak) %in% Oak_species$SpeciesCode]
Oak_explan <- Oak[, ! colnames(Oak) %in% Oak_species$SpeciesCode]
```

See the 'Loading Data' chapter if you do not understand what these actions accomplished.

Transforming the Response Variables

Applying a transformation to an object automatically applies it to each element within the object. Let's apply a square root transformation to our response variables:

```
Sqrt_Oak_abund <- sqrt(Oak_abund)
```

Compare the two objects to verify that the data changed as intended.

The corresponding value in `Sqrt_Oak_abund` of any value in `Oak_abund` can be calculated using the function that we applied to the object – this is one way to see that this was a transformation rather than a relativization.

Transforming Explanatory Variables

There are many types of explanatory variables – continuously distributed predictors, experimental factors, etc. It therefore would generally not make sense to apply the same transformation to a matrix of explanatory variables.

It does make sense to transform individual variables. Each explanatory variable can be evaluated separately to determine which type of transformation, if any, is appropriate.

Let's transform the number of large oak trees. This is a count, so we'll use a log transformation:


```
Oak_explan$log_Quga <- log10(Oak_explan$Quga.gt60cm + 1)
```

I added one to all values to account for the possibility that a stand may not have had any large oak trees.

In fact, this variable is already present in the data frame as the variable 'LogQuga.gt60cm'. Compare these two variables to verify that our calculation was done correctly. The existing variable is reported to two decimal places so we'll round our variable the same:

```
Oak_explan$log_Quga <- round(Oak_explan$log_Quga, 2)
rownames(Oak_explan[which(Oak_explan$log_Quga != Oak_explan$LogQuga.gt60cm), ])
```

Verify the outcome of the above comparison by changing from a test for inequality (!=) to a test for equality (==).

Concluding Thoughts

Decisions about whether and how to transform the data can strongly affect the conclusions of subsequent analyses. Most of the techniques that we are using in this course make minimal statistical assumptions, which means that adjustments do not have to be made for statistical reasons but rather can focus on the ecological questions of interest.

Transformations can be applied to both response variables and explanatory variables. Response variables are often transformed *en masse*, while explanatory variables are transformed individually. Each explanatory variable can be evaluated separately to determine which type of transformation, if any, is appropriate.

Transformations should be scripted rather than permanently changing the raw data file. Scripting ensures flexibility to try other adjustments, skip them entirely, etc.

The transformations that have been discussed here are for continuously distributed variables. For categorical explanatory variables, other actions may be required such as combining similar categories together or restricting analyses to focus on a subset of the categories. These decisions should be based on the objectives of the analysis and the ecological questions that you seek to answer.

References

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Warton, D.I., and F.K. Hui. 2011. The arcsine is asinine: the analysis of proportions in ecology. *Ecology* 92:3-10.

7. Relativizations

Learning Objectives

- To consider how relativizations / standardizations relate to the research questions being addressed.
- To illustrate how to relativize data in R.

Readings

Fox (2013 blogpost)

Key Packages

```
require(vegan)
```

Relativizations / Standardizations

A relativization or standardization is a transformation that is **affected by other elements** in the matrix. Another way to think of this is that the action taken on an individual element would be different if you consider it alone or as part of a set of elements.

Relativizations can be applied to rows (sample units), columns (variables), or both. For example, relativizing a data frame by columns will mean that each element within a given column is adjusted based on the values of other elements in that same column.

Relativizations can also be applied in series – for example, relativizing elements on a row-by-row basis and then further relativizing those relativized elements on a column-by-column basis. See the Wisconsin standardization below as one example of this.

Relativizations are particularly important in some situations:

- To permit equitable comparisons when variables are measured in **different units**. For example, the metadata of our sample dataset indicates that tree abundance is expressed as basal area

(ft²/acre) whereas the abundance of other taxa is expressed as percent cover. Furthermore, note that basal area has no upper bound whereas percent cover cannot exceed 100% for a species. It therefore would not make sense to directly compare the absolute abundance of a tree species to the absolute abundance of another type of species.

- To permit equitable comparisons when variables are expressed on **different scales**. Imagine a study in which the heights of trees and grasses were measured. The trees might be measured in m and the grasses in cm. Simply expressing them on the same scale (e.g., cm) would mean that the much larger numerical values for trees would overwhelm the smaller value for grasses.
- To explore questions about **relative differences**, as described below.

The relativizations described below are often based on a single variable, but another approach is to relativize on the basis of another variable. For example, in wildlife studies abundance is commonly divided by sampling effort to account for differences in sampling effort (Hopkins & Kennedy 2004). This was also evident in the medical trial reported below, where the number of cases was expressed per 10,000 women.

We'll talk about distance measures soon, but for now I'll note that some are based on absolute differences between sample units and others on relative differences between sample units. Many distance measures based on absolute differences become mathematically equivalent following relativization.

Relativizations

Relativizations are conditional: the change in one element depends on which other elements are present in the object.

Theory

Analyses based on absolute values and on relative values address different questions. Imagine a study in which you capture and count the animals in an area. What hypotheses might be tested if you analyze the number of raccoons captured (an absolute value)? What about if you analyze the proportion of all animals captured that were raccoons (a relative value)?

The below image nicely illustrates the ways that absolute and relative effects can relate to one another, including that one effect can be a cost (negative) while the other is a benefit (positive)!

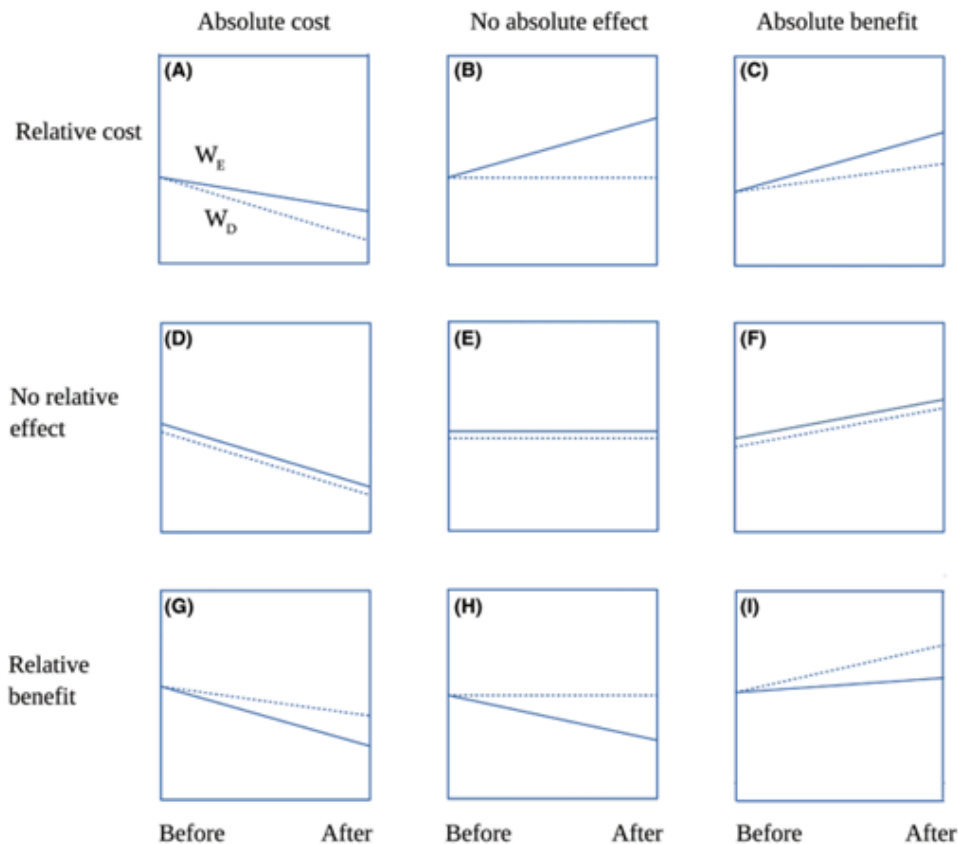


Figure 1. Schematic presentation of the performance (W) of a plant before and after a given period of time, in which the plant either stayed emergent (E) or went into vegetative dormancy (D). The environment is assumed to be constant. An absolute benefit or cost of dormancy refers to an increase or decrease, respectively, in plant performance after dormancy compared to the initial state. A relative benefit or cost refers respectively to improved or decreased performance of a plant following dormancy as compared to a scenario in which it stays emergent (see also [Supporting Information part 1](#)). Panels A–I present the nine possible combinations of the presence and direction of the absolute and relative effects of dormancy: A, dormancy has both absolute and relative costs; B, dormancy has relative cost, but no absolute effect; C, dormancy has absolute benefit but relative cost; D, dormancy has an absolute cost, but no relative effect; E, dormancy has no absolute or relative effect; F, dormancy has an absolute benefit and no relative effect; G, dormancy has a relative benefit and no absolute effect; H, dormancy has absolute cost, but relative benefit and

Examples of how absolute and relative effects can have different expectations. These examples are from a study of plants that were assessed before and after they either stayed aboveground or went into dormancy. The combinations range from dormancy having (A) a cost in both absolute and relative terms, (B) absolute benefit but relative cost, (G) absolute cost but relative benefit, and (I) benefits in both absolute and relative terms. From Hurskainen et al (2018).

Grace (1995) provides an example of how a dataset can be used to answer one set of questions based on absolute values and another set of questions based on relative values (see Freckleton et al. (2009) for more of this conversation). And, Fox (2013) provides some nice examples of the difference between absolute and relative fitness.

The fact that absolute and relative values can answer different questions can be used to mislead. As an example of the latter, Yates (2019) notes that medical trials commonly report positive outcomes “in relative terms, to maximize their perceived benefit” but side effects “in absolute terms in an attempt to minimize the appearance of their risk” (p. 133). He illustrates this with a study of a breast cancer treatment, in which the drug reduced the occurrence of breast cancer but was also associated with increased occurrence of uterine cancer. I’ve summarized the data below, **highlighting** the statistics that were used to interpret the results. Can you see how this can be misleading?

Treatment	Cases Per 10,000 Women	
	Breast Cancer	Uterine Cancer
Drug	133	23
Placebo	261	9
Relative Change due to Drug (difference between Drug and Placebo, divided by Placebo)	-49%	156%

How to decide what to do? As McCune & Grace (2002, p. 70) note, **“there is no right or wrong answer to the question of whether to relativize until one specifies the question and examines the properties of the data”**.

Absolute and Relative Effects

Knowing the direction and magnitude of an absolute effect does not necessarily tell you the direction and magnitude of a corresponding relative effect. It is possible for them to respond similarly, or for one to be positive and the other to be neutral or negative.

Types of Relativizations

I summarize some common relativizations here.

Normalize (Adjust to Standard Deviate)

If variables are normally distributed, they can be standardized into Z-scores:

$$Z = \frac{Y_i - \bar{Y}}{s}$$

where Y_i is the value of Y in the i^{th} sample unit, \bar{Y} is the mean value of Y , and s is the standard deviation of Y .

After normalization, the values for a variable are expressed in units of how many standard deviations they are from the mean (negative if below the mean, positive if above the mean). As a result, this relativization can permit equitable comparisons among variables even if they were measured in very different units.

This is **commonly applied to columns of normally-distributed data**. This relativization is generally not applied to species abundances but is very reasonable for other response variables and for some explanatory variables.

It rarely makes sense to apply this relativization to rows.

Relativize by Maximum

Set maximum value to 1 and calculate all other elements as a proportion between 0 and 1. This is **commonly applied to the columns (species) of a species abundance matrix** or to other variables where zero is a reasonable expectation for a minimum value.

When applied to species abundance data, this allows species to contribute equally to differences between plots. Assuming that two species were absent from some plots, and thus have zero as their minimum, their relativized values will span the same range even if they differed greatly in abundance.

Relativize by Range

Set maximum value to 1 and minimum value to 0, and calculate all other elements as proportions between these two values.

This relativization is similar to relativizing by maxima but is useful for variables that do not have zero as the expected minimum value. For example, it would be more appropriate to relativize the latitude of each plot (`Oak$LatAppx` or `Oak_explan$LatAppx`) by range than by maxima. Do you see why this is the case? Try calculating and comparing the two approaches!

Adjust to Mean

Subtract row or column mean from each element. An element that was smaller than the mean will produce a negative number, while an element that was larger than the mean will produce a positive number.

This is the numerator of the Z-score calculation used to normalize data above. However, it does not change the units in which variables are measured.

Binary

If binary decisions are made on the basis of a static value, such as 'greater than zero', then they are transformations as discussed in that chapter. However, if the threshold value depends on other elements in the matrix, then a binary adjustment is a relativization. For example, one plausible relativization is to set all values smaller than the median to 0 and all values larger than the median to 1. Since the median depends on other values in the matrix, the outcome for any element depends on which other elements are included in the dataset.

Weight by Ubiquity / Relativize by Total

Calculate value in each element as a proportion of the total of all values.

This is **commonly applied to the rows (sample units) of a species abundance matrix**. The resulting data equalizes the contribution of plots: the relativized data sum to 1 for every plot regardless of how much plots differed in total abundance. Note that it only makes sense to apply this to a set of values where the sum of those values is meaningful.

Sometimes it makes sense to apply this to a subset of the variables. For example, if the explanatory variables include depths of different soil horizons, you could relativize each horizon as a proportion of the total soil depth.

It rarely makes sense to apply this relativization to columns. To do so, the sum of the values in the column would have to be a meaningful value.

Common Relativizations

The most common relativizations are:

- Relativizing each column by its maximum or range or, if normally distributed, by converting it to a Z-score. These approaches equalize the importance of each column by accounting for differences in unit, scale, etc.
- Relativizing each row by its total. This equalizes the contribution of each row by adjusting for differences in total.

Will Relativization Make a Difference?

McCune & Grace (2002, p.70) note that the degree of variability in row or column totals can be used to assess whether relativization will have much of an effect. They are referring here to data that have been measured in the same units for all variables (columns) in all sample units (rows). In other words, it needs to make sense to compare row or column totals.

McCune & Grace express the degree of variability by the coefficient of variation (CV) of the row or column totals. The CV is the ratio of the standard deviation of a variable to its mean, multiplied by 100 to express it as a percentage. They propose the following benchmarks:

CV (%)	Magnitude
< 50	Small
50-100	Moderate
100-300	Large
> 300	Very large

The predicted effect of relativization is directly related to the magnitude of the CV: relativization by columns will have a greater effect if there is very large variation among the columns than if there is small variation among them. However, please note that these are only rules of thumb. Depending on your objectives, it may make sense to relativize even if the CV is small or to not relativize even if the CV is very large.

Row and column totals can be calculated using the `rowSums()` and `colSums()` functions. We can use these to calculate the CV.

We'll illustrate this with our oak plant community dataset. To begin, we open the R project, load the data, and create separate objects for the response and explanatory data:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
Oak_abund <- Oak[, colnames(Oak) %in% Oak_species$SpeciesCode]
Oak_explan <- Oak[, ! colnames(Oak) %in% Oak_species$SpeciesCode]
```

See the 'Loading Data' chapter if you do not understand what these actions accomplished.

Now, let's calculate the CV among row totals:

```
100 * sd(rowSums(Oak_abund)) / mean(rowSums(Oak_abund)) # CV of row (plot) totals
```

Aside: A Simple Function to Calculate Coefficient of Variation

Note that we call `rowSums(Oak_abund)` twice in the above calculation of the CV. Adapting this for a different variable can therefore be error prone – for example, if we want to apply it to columns we might accidentally forget to change one of these calls to `colSums(Oak_abund)`. Importantly, the code would still execute but the resulting numbers would be nonsensical.

An alternative is to create a function in which the repeated object is only specified once. To do so, we will replace the repeated object by a single argument:

```
CV <- function(x) { 100 * sd(x) / mean(x) }
```

This provides an opportunity to highlight some basic points about writing functions:

- Functions are created using the `function()` function, and are assigned to an object name – in this case, 'CV'.
- The `function()` function should include any arguments that you want to be able to specify while executing the function. In this case, we only have the argument `x`, which is the object for which we want to calculate the standard deviation and mean. Within the function, we specify this argument wherever we want it to be replaced by the specified object when the function is executed.
- The `function()` function can also include any arguments that we want to be able to specify but for which we also want predefined default values. This example doesn't have any such arguments, but they would take the form 'argument = default'.
- The {squiggly brackets} denote the beginning and end of the function; in this case it is only one line long. Other functions can span tens or hundreds of lines of code.
- Functions are assigned to an object name, and called using that name.

Applying our new `CV()` function:

```
CV(x = rowSums(Oak_abund)) # CV of plot totals
CV(x = colSums(Oak_abund)) # CV of species totals
```

This function can be used with any data, not just the row and column sums that we are focusing on here. For example:

```
CV(x = Oak_explan$Elev.m) # CV of plot elevations
```


Again, the advantage of a function is that all of the operations specified in the function are carried out every time it is called.

If you include the code for a function near the beginning of your script, it will be loaded each time you run your script, and available for use later in the script.

Applications in R

In R, there are several functions that can be used to relativize data. Some are applied to rows or columns by default – these are generally reasonable defaults, but necessary to be aware of.

apply()

The `apply()` function can be used to apply many functions to a matrix. The usage of this function is:
`apply(X, MARGIN, FUN, ..., simplify = TRUE)`

The key arguments are:

- `x` – the matrix to be analyzed
- `MARGIN` – whether to apply the function to rows (1), columns (2), or both (`c(1,2)`)
- `FUN` – the function to be applied

For example, we could use this function to calculate the species richness of each stand:
`apply(X = oak_abund > 0, MARGIN = 1, FUN = sum)`

Note that we have incorporated some indexing into this function. Do you understand what this indexing did?

What would we be calculating if we switched the margin from 1 to 2?

scale() and sweep()

The `scale()` function centers and/or scales (i.e., normalizes) a matrix. Its usage is:
`scale(x, center = TRUE, scale = TRUE)`

The key arguments are:

- `x` – the data matrix
- `center` – Subtract a specified value from each element in a column
 - `center = TRUE` – subtract column mean. This is the default.
 - `center = FALSE` – no centering done
- `scale` – divide each element in a column by a value
 - `scale = TRUE` – divide by root mean square (standard deviation). This is the default.
 - `scale = FALSE` – no scaling done

Note that centering always precedes scaling. If the default values are accepted, this normalizes each column.

In comparison, the `sweep()` function adjusts data on the basis of a summary statistic. This sounds and is generic – centering is a specific example in which the data are adjusted by subtracting the mean value. No scaling is involved.

For example, imagine that we want to express the abundance of species relative to their median values. You can verify that the following functions are identical:

```
Quga <- Oak_abund[ , c("Quga.s", "Quga.t")]
scale(x = Quga, center = apply(Quga, 2, median), scale = FALSE)
sweep(Quga, 2, apply(Quga, 2, median))
```

(note that I restricted attention here to the two oak taxa simply for convenience when comparing them visually. Both functions could equivalently be applied to the entire data matrix).

vegan::decostand()

The `decostand()` function is a versatile means of relativizing (standardizing) data. Its usage is: `decostand(x, method, MARGIN, range.global, logbase = 2, na.rm = FALSE, ...)`

The key arguments are:

- **x** – the data frame or matrix to be relativized
- **method** – relativization / standardization method to be applied. Each method has a default margin but can also be applied to the other one. I've highlighted those that are particularly useful as noted above.
 - **total** – divide by margin total. By default, applied to rows.
 - **max** – divide by margin maximum. This equalizes the contribution of each species in the matrix. By default, applied to columns.
 - **frequency** – divide by margin maximum and then multiple by the number of non-zero elements so that the average of the non-zero entries is one. By default, applied to columns.
 - **normalize** – make margin sum of squares equal to one. By default, applied to rows. Note that this is not the same as the normalizing that is discussed above.
 - **range** – convert values to range from 0 to 1. By default, applied to columns.
 - **rank** – convert abundance values to increasing ranks – in other words, the most abundant species will have the largest rank. Zeroes are left unchanged. By default, applied to rows.
 - **rrank** – rank, relativized so that the highest rank is 1. By default, applied to rows.
 - **standardize** – normalize data by converting it to Z-scores (i.e., mean = 0, variance = 1). Intended for normally distributed data. By default, applied to columns.
 - **pa** – convert to presence/absence (1 or 0, respectively).
 - **chi.square** – Divide by row sums and square root of column sums, and adjust for square root of matrix total. Can be problematic; we'll discuss this in the context of correspondence analysis. By default, applied to rows.
 - **hellinger** – square root of **total** method. By default, applied to rows.
 - **log** – adjusts values based on a log (specified by **logbase**). Values greater than zero are log-transformed, while zeroes are left unchanged. Note, therefore, that this is not a simple log transformation; see help for details.
 - **alr** – additive log ratio. Commonly used with pH and other chemistry measurements.
 - **clr** – centered log ratio. Commonly used in microbial ecology.
 - **rclr** – robust centered log ratio.
- **MARGIN** – whether to apply to rows (**MARGIN** = 1) or columns (**MARGIN** = 2). Each method has a

- default margin – explained in the ‘Details’ section of the help file – so this only has to be included if you want to apply the method to the non-default margin.
- `na.rm` – whether to ignore missing values

As an example, let's relativize elevations by their range:

```
decostand(x = Oak_explan$Elev.m, method = "range")
```

In this case, the relativization was applied to one variable. Applying it to the entire dataframe is simply a matter of indexing, and of course making sure that `decostand()` is being applied to the correct margin. If we want to relativize our response data by the maximum of each species (column):

```
Oak_abund_max <- decostand(x = Oak_abund, method = "max")
```

vegan::wisconsin()

When working with species abundance data, a common approach is to standardize each species by its maximum, and then each site by its total. This double standardization is known as a Wisconsin standardization as it was first applied to data collected in that state.

Wisconsin standardization is simply done using the `wisconsin()` function:

```
Oak_abund_stand1 <- wisconsin(Oak_abund)
```

Verify that this is identical to the result of two calls to `decostand()`, first standardizing columns by their maxima and then rows by their totals:

```
Oak_abund_stand2 <- decostand(decostand(Oak_abund, "max"), "total") # nested functions
Oak_abund_stand2 <- Oak_abund |> decostand("max") |> decostand("total") # piped
```

Concluding Thoughts

Decisions about how to adjust data – including whether and which variables to relativize – can strongly affect the conclusions of subsequent analyses. Relativizations can be made to both response variables and explanatory variables. Most of the techniques that we are using in this course make minimal statistical assumptions, which means that adjustments do not have to be made for statistical reasons but rather can focus on the ecological questions of interest.

With species data, several types of standardization are common, depending on the objectives of the study. Some studies will not do any standardization, some will standardize columns only, some will standardize rows only, and some will standardize both rows and columns.

Relativizing by row may be appropriate for a full matrix or for appropriate subsets of the data. For example, it would not be appropriate to relativize by rows if the columns of interest were not logically related to one another (e.g., elevation, soil depth, aspect).

Relativizing by column can be particularly helpful when variables have not been measured on the same scale but want to be given equal weight. This is true for both response and explanatory variables.

Actions such as deleting rare species, transforming data, and relativizing data should be done

through scripts, not through permanent changes to the raw data file. Scripting these actions ensures flexibility to try other adjustments, skip them entirely, etc.

Sometimes answering our ecological questions require a series of adjustments. The order of adjustments can affect the resulting data. See this appendix for guidance about the order in which to conduct data adjustments. For example, in one study (Mitchell et al. 2017) we analyzed plant community structure. Our raw data was a plot x species matrix where the elements were the abundance of each species in each plot in 2002. Our workflow included a number of steps:

- Summed the abundances of all species on a row-by-row basis to yield total cover.
- Transformed the abundances to presence/absence data, and then summed those data on a row-by-row basis to yield species richness.
- Grouped species by their functional group, and calculated the cover for each functional group. Then, we relativized the abundances by row totals (so, each row summed to 1), and calculated the proportion of the cover accounted for by each functional group. Thus, we were focusing on the relative abundance of these functional groups. We did not analyze absolute abundances.
- Combined these data with a similar plot x species matrix of the same plots in 1989 to determine how much composition had changed during this period. To combine data from these two years, we had to resolve taxonomic differences between years. We also elected to delete rare species. After these adjustments, we calculated the compositional change from 1989 to 2002 as the Bray-Curtis dissimilarity (one of the distance measures we'll discuss soon) between the two years.

Which Adjustments to Make, and in What Order?

A series of adjustments are often required to organize the data in a way that permits our questions to be answered. Think carefully through which adjustments to make and in which order to make them – these decisions can affect the conclusions of analyses.

References

- Fox, J. 2013. Do ecologists ever confuse absolute and relative fitness? <https://dynamicecology.wordpress.com/2013/09/24/dont-confuse-absolute-and-relative-fitness/>
- Freckleton, R.P., A.R. Watkinson, and M. Rees. 2009. Measuring the importance of competition in plant communities. *Journal of Ecology* 97:379-384.
- Grace, J.B. 1995. On the measurement of plant competition intensity. *Ecology* 76:305-308.
- Hopkins, H.L., and M.L. Kennedy. 2004. An assessment of indices of relative and absolute abundance for monitoring populations of small mammals. *Wildlife Society Bulletin* 32:1289-1296.
- Hurskainen, S., K. Alahuhta, H. Hens, A. Jäkäläniemi, T. Kull, R.P. Shefferson, and J. Tuomi. 2018. Vegetative dormancy in orchids incurs absolute and relative demographic costs in large but not in small plants. *Botanical Journal of the Linnean Society* 188:426-437.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Mitchell, R.M., J.D. Bakker, J.B. Vincent, and G.M. Davies. 2017. Relative importance of abiotic, biotic, and disturbance drivers of plant community structure in the sagebrush steppe. *Ecological Applications* 27:756-768.

Yates, K. 2019. *The Math of Life & Death: 7 Mathematical Principles That Shape Our Lives*. Scribner, New York, NY.

Media Attributions

- Hurskainen.et.al.2018_Figure1

8. Matrix Algebra Basics

Learning Objectives

To describe the basics of matrix algebra.

To continue using R.

Resources

Gotelli & Ellison (2004, appendix)

Introduction

This chapter takes a detour to cover some key mathematical concepts that are foundational to everything that follows in this course. Some of this will seem abstract initially, but it's followed by a worked regression example and an introduction to eigenanalysis which forms the basis for principal component analysis (PCA)!

By convention, mathematicians use bold uppercase text (**X**) to indicate a matrix and bold lowercase text (**x**) to indicate a vector. However, these conventions don't apply in R: bold formatting isn't available, and we can name a matrix object like we name any other object.

Vectors

Recall that a **scalar** is a single numerical value:

```
x <- 5
```

A **vector** is a series of n scalars. Mathematically, a vector is assumed to be a column vector (i.e., n rows x 1 column), though see below for some nuance about this.

Vectors can be created in several ways. One way is to combine a series of values using the `c()` function (note that this is a lower-case `c`):

```
v <- c(1,3,5,7,9); v
```

```
1 3 5 7 9
```

Another way is by specifying a sequence of values:

```
v <- seq(from = 1, to = 9, by = 2); v # Equivalent  
v <- seq(1, 9, 2) #same, without argument names
```

These commands are extremely versatile.

Vectors can be indexed using `[]` just like we do for a matrix. However, a matrix is indexed with respect to `[row, column]`, while a vector just has to be indexed with respect to the position of an element within the vector:

```
v[3]
```

Nuances About Vectors in R

R displays vectors as row vectors (i.e., 1 row x n columns) as this takes less screen space than a column vector (i.e., n rows x 1 column). See the output of `v` above for an example of this. Mathematically, however, vectors are always assumed to be column vectors (n x 1).

R attempts to make all operations work: “if you use a vector in an operation that succeeds or fails depending on the vector’s orientation, **R will assume that you want the operation to succeed and will proceed as if the vector has the necessary orientation**” (Ellner & Guckenheimer 2011, p. 13). This can cause problems because an operation can work even when you don’t want it to, or where it should not mathematically!

Matrices

A matrix is a two-dimensional object consisting of m vectors. Since each vector is of length n , the size of the matrix is n x m . Note that all vectors have to be the same length.

The `matrix()` function can be used to create a matrix from a sequence of numbers. The `nrow` argument tells R how many rows to include.

```
A <- matrix(data = c(3,-1,0,4,5,2), nrow = 3); A
```

```
      [,1] [,2]  
[1,]    3    4  
[2,]   -1    5  
[3,]    0    2
```

Alternately, you could use the `ncol` argument to specify how many columns to include. You don't need to specify both because R determines the other dimension based on the number of elements included in the data argument. If you specify a number of rows (or columns) that is not an integer divisor of the total number of cells, R will display a warning message and recycle the numbers to complete the matrix:

```
A1 <- matrix(data = 1:9, nrow = 4); A1
```

The `matrix()` function includes other arguments besides `nrow` and `ncol`. The arguments are described in the R help:

```
?matrix
```

Note the default values for each argument. For example, the `byrow` argument defaults to `FALSE`, meaning that data are not added one row at a time but rather are added on a column-by-column basis. If you set `byrow = TRUE`, data are added on a row-by-row basis:

```
B <- matrix(data = c(3,-1,0,4,5,2), nrow = 3, byrow = TRUE); B
```

Compare objects `A` and `B` to verify that the data have been added to the matrix in different orders.

Recall that we use square brackets to index elements in a matrix (or data frame), and we always discuss elements relative to `[row, column]`. This indexing enables you to call the elements that you're interested in. These elements can be viewed, assigned to another object name, or manipulated:

```
A[3, 1]
C <- B[1:2, ]; C
B[1, ] <- B[1, ] * 10; B # What does this do?
B[B > 2] # And this?
```

The size of a matrix is important for operations such as matrix addition, subtraction, and multiplication. Its size is expressed by its dimensions, which can be obtained using the `dim()` function, or by requesting the number of rows or columns specifically:

```
dim(A)
nrow(A)
ncol(A)
```

Although a vector can mathematically be considered a matrix of $n \times 1$ dimensions, as noted above, these functions don't work for vectors; use `length()` instead. If you want to force a vector to be recognized as a matrix, you have to set it to that class.

Matrices can be combined with respect to rows or columns:

```
rbind(A,B) # stacked
cbind(A,B) # side-by-side
```

However, matrices can only be combined if the appropriate dimension is the same for each. For example, two matrices that have the same number of rows (and any number of columns) can be combined with respect to columns (i.e., side-by-side):

```
cbind(A, B)
cbind(A, C) # Doesn't work!
```

The number of columns in each matrix doesn't affect `cbind()`. However, the opposite is true with

`rbind()`: the matrices must have the same number of columns, and the number of rows doesn't matter.

Keep in mind that there must be a logical reason to combine the matrices in a particular order. If **A** and **B** are plot x species matrices, what are we assuming when we combine them using `cbind()`? In addition, with `rbind()` note that the columns that are being stacked together must contain the same class of data.

Matrix Addition and Subtraction

Matrix addition and subtraction are performed **element-by-element**. Therefore, the matrices to be added or subtracted must be of identical dimensions – both rows and columns. The number of rows does not have to equal the number of columns, however.

To demonstrate this, we'll begin by re-assigning **A**, **B**, and **C** to correspond to Equation A.1 from Gotelli & Ellison (2004):

```
A <- matrix(data = c(3,-1,0,4,5,2), nrow = 3)
B <- matrix(data = c(4,1,-2,8,-3,6), nrow = 3)
C <- A + B; C
```

Matrices are commutative with respect to addition – the order of the terms doesn't matter:

```
A + B
B + A
```

Matrices are also associative with respect to addition – the order in which the operations are conducted, as determined by which terms are in parentheses, doesn't matter:

```
(A + B) + C
A + (B + C)
```

However, matrices are **not** commutative or associative with respect to subtraction (Gotelli & Ellison are incorrect in the first edition of their book; I haven't checked if this has been fixed in later editions):

```
A - B
B - A
```

```
(A - B) - C
A - (B - C)
```

Scalar and Matrix Multiplication

Two types of multiplication are possible with vectors and matrices: element-by-element and matrix.

Scalar Multiplication

Element-by-element multiplication is the standard type of multiplication (indicated by `*`) and is the default in R. This is most easily seen by multiplying a matrix by a scalar (single value). Each element in the matrix is simply multiplied by the scalar. This process is commutative, associative, and distributive. For example, here is Equation A.2 from Gotelli & Ellison (2004):

```

k <- 3
A <- matrix(data = c(1,2,3,0,2,0,-5,3), nrow = 2)
k * A
A * k

c <- 5
c * (k * A)
(c * k) * A

B <- matrix(data = c(1,2,3,0,2,0,-5,3), nrow = 2)
k * (A + B)
k * A + k * B

```

This type of multiplication can even be applied to matrices, if those matrices are of the same size. The value in one position in the matrix is simply multiplied by the value in the same position in the other matrix. For example:

```
A * A
```

View this file to verify that we have simply squared every value.

Matrix Multiplication

Matrix multiplication is a bit more complicated than scalar multiplication – in fact, it is not always possible. **The number of columns in the first matrix must equal the number of rows in the second matrix** – if they are not equal, matrix multiplication cannot occur.

One way to figure out whether two matrices can be matrix multiplied is by writing down the dimensions of each matrix. For example, here are the matrices from Equation A.3 of Gotelli & Ellison (2004):

```

A <- matrix(data = c(1,2,3,0,3,-2), nrow = 2) # A is a 2x3 matrix
B <- matrix(data = c(1,3,-1,-2,2,3,2,2,3), nrow = 3) # B is a 3x3 matrix

```

Matrix multiplying **A** and **B** involves combining a (2×3) matrix and a (3×3) matrix.

- Since the inside numbers are the same, they can be matrix multiplied.
- The outside dimensions indicate the dimensions of the resulting matrix (here, 2 rows x 3 columns).

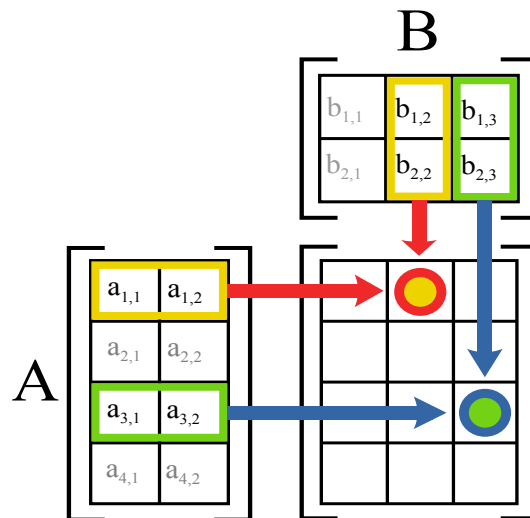
Why? Well, it relates to how the matrix multiplication is actually conducted. Recall that we always refer to rows first, columns second. In matrix multiplication, each element in a row of the first matrix is multiplied by the element in the corresponding column and position of the second matrix, and the products are summed together. If the inner dimensions of the two matrices aren't equal, there will be elements in one matrix that cannot be multiplied by something in the other matrix.

The calculation for the first row of **A** and first column of **B** in our example is:

Element locations: $([1,1] * [1,1]) + ([1,2] * [2,1]) + ([1,3] * [3,1])$

Data: $(1 * 1) + (3 * 3) + (3 * -1) = 7$

This process is illustrated graphically for two values below.



Example of how matrix multiplication is the sum of the products of the elements in a row of one matrix (A) and the elements in a column of another matrix (B). From Wikipedia.

The matrix multiplication symbol (`%*%`) tells R to repeat this calculation for each pair of elements in the matrices:

```
A %*% B
```

Assuming the matrices are of the proper dimensions to actually be multiplied, matrix multiplication is associative:

```
A %*% (B %*% B)
```

```
(A %*% B) %*% B
```

However, it is not commutative:

```
A %*% B
```

```
B %*% A # Doesn't work
```

Matrix Multiplication

The order of the matrices matters during matrix multiplication. As a result, the terminology needs to be precise. For example, the equation **AB** can be equivalently described as:

- The premultiplication of **B** by **A**
- The postmultiplication of **A** by **B**

Matrix Transposition and Symmetry

In some cases, matrix multiplication is not possible in one configuration but would be possible if the rows and columns of one of the matrices were reversed. This reversal is known as **transposition**, and is easily done in R:

```
A      # 2x3 matrix
```

```
t(A)   # 3x2 matrix
```

The transpose of **A** is written as **A'** or **A^T**.

We saw above that **BA** did not work, but compare this with **BA^T**:

```
B %% t(A)      # Why does this work?
```

Note that the dimensionality of **BA^T**, and the elements within this product, differ from those in **AB**.

Many matrices are **square** – the two dimensions are the same. Square matrices can have many desirable properties. In particular, square matrices can be **symmetric**, meaning that they are identical to their transpose. This can be verified by visually comparing a matrix with its transpose or by subtracting one from the other (if symmetric, what should the result be?).

```
A <- matrix(data = c(1,2,2,2,4,3,2,3,-4), nrow = 3) # What are the dimensions of this matrix?
```

```
A
```

```
t(A)
```

```
A - t(A)
```

However, not all square matrices are symmetrical:

```
B <- matrix(data = c(1,3,-1,-2,2,3,2,2,3), nrow = 3)
```

```
B
```

```
t(B)
```

```
B - t(B)
```

We will see many symmetric square matrices (diagonal matrices, identity matrices, variance-covariance matrices, distance matrices, etc.) throughout this course.

The **diagonal** elements of a matrix are often important, and can be easily extracted:

```
diag(x = A)
```

This command can also be used to modify the values on the diagonal of a matrix:

```
A
```

```
diag(x = A) <- 10; A
```

Key Features of (Some) Matrices

Matrices with the following features are particularly useful:

- Square
- Symmetric (identical to their transpose)

All symmetric matrices are square, but not all square matrices are symmetric.

Matrix Inversion

An **identity matrix** is a symmetric square matrix with 1s on the diagonal and 0s elsewhere. It can be created using the `diag()` function. Since an identity matrix is symmetric, the number of rows and columns are the same and you only need to specify one dimension:

```
I <- diag(x = 4); I
```

Identity matrices are particularly important with respect to **matrix inversion**. The inverse of matrix **A** is a matrix that, when matrix multiplied by **A**, yields an identity matrix of the same dimensions. The inverse of **A** is written as \mathbf{A}^{-1} . The `solve()` function can determine the inverse of a matrix:

```
A
solve(A)
A %*% solve(A) # Verify that the product is an identity matrix
(use the round() function if necessary to eliminate unnecessary decimal places)
```

Inversions only exist for square matrices, but not all square matrices have an inverse.

Sometimes, a matrix has to be inverted to yield something other than an identity matrix. We can do this by adding a second argument to the `solve()` function. Suppose we have the equation $\mathbf{AA}^{-1} = \mathbf{B}$ and need to calculate \mathbf{A}^{-1} :

```
A <- matrix(data = c(1,2,2,2,4,3,2,3,-4), nrow = 3)
B <- matrix(data = c(1,2,3,2,3,1,3,2,1), nrow = 3)
A.inv <- solve(A, B)
A %*% A.inv # Verify the result! What should this be?
```

A matrix is **orthogonal** if its inverse and its transpose are identical. An identity matrix is an example of an orthogonal matrix.

Concluding Thoughts

Matrix algebra is an incredibly powerful tool when dealing with multivariate data, and we will be using these concepts throughout the quarter. For example, distance measures convert observations

of m variables on n sample units (i.e., a $n \times m$ data matrix) into a $n \times n$ symmetric square distance matrix.

References

Ellner, S.P., and J. Guckenheimer. 2011. *An introduction to R for dynamic models in biology*. <http://www.cam.cornell.edu/~dmb/DynamicModelsLabsInR.pdf>

Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer Associates, Sunderland, MA.

Media Attributions

- Matrix_multiplication_diagram_2 is licensed under a CC BY-SA (Attribution ShareAlike) license

9. Matrix Algebra to Solve a Linear Regression

Learning Objectives

To illustrate how matrix algebra can solve a linear regression.
To continue using R.

Resources

Gotelli & Ellison (2004, appendix)

Introduction

Matrix algebra is helpful for quickly and efficiently solving systems of linear equations. We will illustrate this by using it to solve a linear regression.

Matrix Formulations of Regression

The matrix algebra formulation of a linear regression works with any number of explanatory variables and thus is incredibly flexible.

Recall that the model for a simple linear regression is $y = mx + b$, where b and m are coefficients for the intercept and slope, respectively. Let's rearrange this slightly and rewrite m as b_1 :

$$y = 1b_0 + xb_1$$

Note that b_0 is equivalent to $1b_0$. In other words, the right-hand side of this equation consists to the sum of two products: 1 times the intercept plus the measured value of x times the slope. We can

now summarize this in matrix form:

$$\mathbf{Y} = \mathbf{X}\mathbf{b}$$

where

- \mathbf{X} is a matrix with as many rows as there are data values and two columns (a column of 1s and a column of x values)
- \mathbf{b} is a vector of two coefficients (intercept and slope)

When we fit a simple linear regression to data, we are determining the coefficients associated with each variable. Equation A.14 from Gotelli & Ellison (2004) states that the coefficients (i.e., \mathbf{b}) can be calculated as

$$\mathbf{b} = [\mathbf{X}^T\mathbf{X}]^{-1}[\mathbf{X}^T\mathbf{Y}]$$

Chlorella Example

Let's use a simple example to see how these calculations work. We will use a dataset containing the maximum per-capita growth rate of an alga, *Chlorella vulgaris* (y), and light intensity (x). These data are from Ellner & Guckenheimer (2011). The dataset is available in CSV format through the book's GitHub site. Download it into the 'data' sub-folder within your SEFS 502 folder. Open the course R Project and then read the dataset into R:

```
chlorella <- read.csv("data/chlorella.csv", header = TRUE, row.names = 1)
```

```
head(chlorella)
```

```
      x    y
1 20 1.73
2 20 1.65
3 20 2.02
4 20 1.89
5 21 2.61
6 24 1.36
```

The column y is the vector of responses. The column x is the vector of values for the explanatory variable, light intensity.

To organize the linear regression model in matrix form, we need to combine each value of `chlorella$x` with a '1' that can be multiplied by the intercept b_0 . We'll organize the ones in a column, combine the two columns of explanatory variables, and convert the resulting object to class matrix:

```
X <- matrix(
  data = c(rep(1,11), chlorella$x),
  nrow = 11)
```

Note that `rep(1,11)` simply repeats the number 1 eleven times.

Now we can solve equation A.14:

```
b <- solve(t(X) %*% X) %*% (t(X) %*% chlorella$y)
```

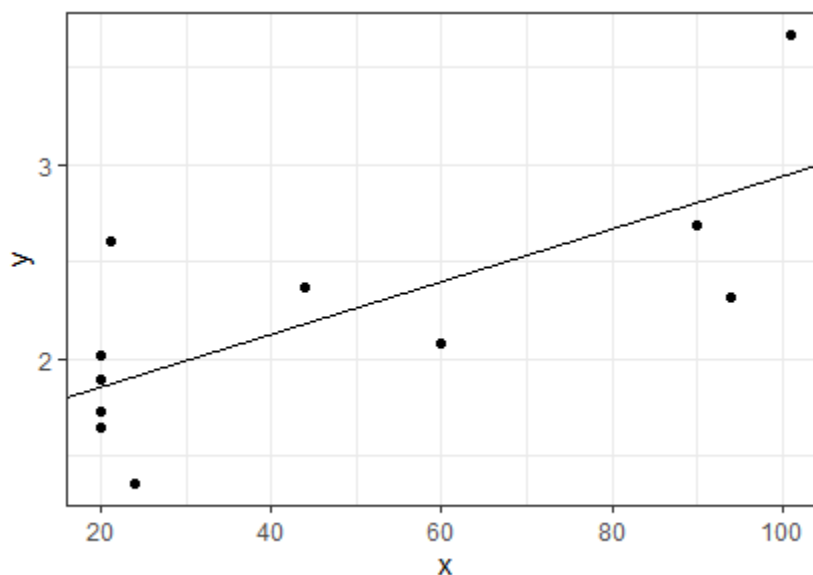
I have simply restated the equation from above, using transposes and inversions as discussed in the 'Matrix Algebra Basics' chapter.

The result of these calculations, the object `b`, is an object that contains the intercept and slope of the equation relating *Chlorella* growth rate to light intensity.

```
      [,1]  
[1,] 1.58095214  
[2,] 0.01361776
```

We can graph the *Chlorella* data and add to it the line described by this slope and intercept:

```
library(ggplot2)  
ggplot(data = chlorella, aes(x = x, y = y)) +  
  geom_point() +  
  geom_abline(intercept = b[1], slope = b[2]) +  
  theme_bw()
```



Relationship between light intensity (x) and Chlorella growth rate (y), as calculated via matrix algebra.

Verification

We can verify our results by comparing them with the coefficients produced from the `lm()` (linear model) function:

```
lm(y ~ x, data = chlorella)
```

In this formula, the `~` means 'as a function of'; here, we fit **y** as a function of **x**. The 1's that are multiplied by the intercept are automatically accounted for when using `lm()` and related functions.

The coefficients are labelled here, and are identical to those that we calculated above.

```
Call:
lm(formula = y ~ x, data = chlorella)
```

```
Coefficients:
(Intercept)          x
  1.58095      0.01362
```

Extensions

Matrix algebra can also be used to calculate other types of information that can be extracted from a regression. Let's walk through a few.

Predicted Values of y

The predicted value of y for each observation (row) is the value obtained by applying the coefficients obtained above to that observation's value of x . In other words, this is the solution to the linear regression formula that we re-arranged above, $\mathbf{Y} = \mathbf{Xb}$:

```
Y_pred <- X %*% b
```

For verification, multiply a value of x times the slope, and add the intercept.

Residuals

The residual for each observation (row) is the difference between its actual and predicted values of y :

```
residuals <- chlorella$y - Y_pred
```

For verification, see `resid(lm.res)`.

Predicted Values for a Range of x Values

The predicted values across a range of x values simply requires that we specify which values of x we want to use. Let's use 50 values that span the range of x in our data:

```
X_range <- matrix(
  data = c(rep(1,50), seq(from = min(chlorella$x), to = max(chlorella$x), length.out
= 50)),
  nrow = 50)
```

```
Y_range <- X_range %*% b
```

These could be the values that are graphed as a fit line in ggplot, for example (you can do so and compare to the above graph to verify).

Note that it isn't necessary to use this many values when graphing a linear fit, but a large number of values would be helpful if we had transformed one of our variables and were back-transforming the fit for presentation – more values will show a smoother curve for the resulting non-linear fit.

Concluding Thoughts

The appeal of this matrix formulation of a linear regression is that it can be easily generalized to any number of explanatory variables – fitting a response as a function of two variables simply adds one column to \mathbf{X} and one value to \mathbf{b} , but does not change the matrix form of the equation, or the corresponding calculation. This gives the matrix formulation of this equation incredible flexibility.

References

Ellner, S.P., and J. Guckenheimer. 2011. *An introduction to R for dynamic models in biology*. <http://www.cam.cornell.edu/~dmb/DynamicModelsLabsInR.pdf>

Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer Associates, Sunderland, MA.

Media Attributions

- chlorella

10. Eigenanalysis

Learning Objectives

To use matrix algebra to conduct eigenanalysis, and to begin interpreting the resulting eigenvectors and eigenvalues.

To continue using R.

Resources

Gotelli & Ellison (2004, appendix)

Introduction

Eigenanalysis is a method of identifying a set of linear equations that summarize a square matrix. It yields a set of **eigenvalues** (λ), each of which has an associated **eigenvector** (\mathbf{x}). The connection between these terms is expressed in Equation A.16 from Gotelli & Ellison:

$$\mathbf{Ax} = \lambda \mathbf{x}$$

In words, this says that the multiplication of a square matrix \mathbf{A} and a vector \mathbf{x} will yield the same values as the multiplication of a scalar value λ and the vector \mathbf{x} . While this may not sound very helpful, it means that data (\mathbf{A}) can be rotated, reflected, stretched, or compressed in coordinate-space by multiplying the individual data points by an eigenvector (\mathbf{x}). We'll see much more about eigenvectors when we discuss ordinations, particularly principal component analysis (PCA).

Eigenanalysis

Eigenanalysis is a method of summarizing a square matrix. Each dimension is represented by an eigenvector and associated eigenvalue.

The dimensions are in decreasing order of importance; each dimension captures as much of the

variation as possible. This is why we can focus on the first few dimensions and be assured that we are seeing the broad patterns within the data.

If all of the eigenvectors are used, the patterns within the data cloud are perfectly preserved even though the cloud itself may be rotated, reflected, stretched, or compressed.

Spectral Decomposition (`eigen()`)

The eigenvalues (λ) and eigenvectors (\mathbf{x}) of a square matrix \mathbf{A} can be calculated using the `eigen()` function:

```
A <- matrix(data = 1:4, nrow = 2)
E <- eigen(x = A); E
```

```
eigen() decomposition
$values
[1]  5.3722813 -0.3722813

$vectors
      [,1]      [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

Note that the eigenvalues and eigenvectors are both reported and can be extracted as necessary for further manipulation:

```
Evalues <- E$values
Eectors <- E$vectors
```

The eigenvalues are always in order of decreasing size. The sum of the eigenvalues is equal to the sum of the diagonal values of the original matrix.

Each eigenvector is associated with the eigenvalue in the same relative position – for example, the first eigenvector (a column) is associated with the first eigenvalue.

Now, let's verify Equation A.16:

```
A %*% Eectors[,1] # Why is this matrix multiplied?
Evalues[1] * Eectors[,1] # Why isn't this?
```

Gotelli & Ellison also state that $(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = \mathbf{0}$ (Equation A.17). Can you verify this equation?

The **trace** of a matrix is equal to the sum of its diagonal values or, equivalently, the sum of its

```
eigenvalues:  
sum(diag(A))  
sum(Evalues)
```

The **determinant** of a matrix is equal to the product of its eigenvalues:
`prod(Evalues)`

It can also be obtained using the `det()` function:
`det(A)`

Singular Value Decomposition (`svd()`)

Another way to obtain the eigenvalues and eigenvectors of a matrix is through singular value decomposition using the `svd()` function:

```
A.svd <- svd(x = A); A.svd
```

```
$d  
[1] 5.4649857 0.3659662  
  
$u  
      [,1]      [,2]  
[1,] -0.5760484 -0.8174156  
[2,] -0.8174156  0.5760484  
  
$v  
      [,1]      [,2]  
[1,] -0.4045536  0.9145143  
[2,] -0.9145143 -0.4045536
```

Gotelli & Ellison (2004) discuss singular value decomposition but use a different symbology in Equation A.22 than is used in the help file associated with `svd()`. The symbology in the R formulation, $\mathbf{X} = \mathbf{UDV}'$, is defined in the table below.

Matrix	Dimension	Notes
X	m x n	The matrix being analyzed
U	m x n	
D	n x n	Diagonal matrix with singular values of X on diagonal
V	n x n	Note: transposed in equation

The output consists of the full **U** and **V** matrices and the singular values of **D**. However, they are referred to using lower-case letters as shown in the above output.

The `eigen()` and `svd()` approaches give broadly similar – but not identical – results:

- `A.svd$d` is analogous to `E$values`. Each of these is an eigenvalue.
- `A.svd$u` is analogous to `E$vectors`. It is a set of vectors, each of which is associated with an eigenvalue and relates to the rows of the original matrix.
- `A.svd$v` is another set of vectors. Each vector is associated with an eigenvalue, but relates to the columns of the original matrix. This is used in Correspondence Analysis (CA), though many people find it problematic. See that chapter for details.

Concluding Thoughts

While eigenvalues and eigenvectors may sound abstract, they are used to rotate, reflect, stretch, or compress data in coordinate-space by multiplying the individual data points by an eigenvector (**x**). The fact that the eigenvectors are in descending order of importance is what allows us to use this as a data reduction approach.

We'll see much more about eigenvectors when we discuss ordinations, particularly principal component analysis (PCA).

References

Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer Associates, Sunderland, MA.

11. Properties of Distance Measures

Learning Objectives

To consider the desirable properties of distance or dissimilarity measures, including the difference between the two.

To introduce the distance matrix as a method of summarizing a set of pairwise distances.

To understand how distance measures use matrix algebra to provide a link between raw data, data adjustments, and techniques to test for statistical differences, identify groups, and visualize patterns.

Introduction

Distance measures are an essential component of many ecological analyses. Here, we'll review the properties of the distance measures that are most commonly used in ecological studies.

Distance measures can be calculated among plots (aka sample units; the rows in your data matrix) or species (aka variables; the columns in your data matrix). However, most analyses are based on the distances among plots, so that's what I'll assume throughout these notes.

Terminology

Many people use the terms '**distance**' and '**dissimilarity**' interchangeably, though some authors recommend using 'distance' only for metric indices (those that satisfy the triangle inequality – see below).

Similarity is the opposite of dissimilarity. Similarity can only be calculated for metrics which have an upper limit.

Simple Examples

A helpful way to begin thinking about the idea of distance is to consider univariate or one-dimensional data. For example, here are data from two plots:

Plot	Total Richness
H	4
I	10

These two plots obviously differ by 6 species. This is the **distance** between the two plots.

Now, suppose that we distinguished annual and perennial plants within each plot:

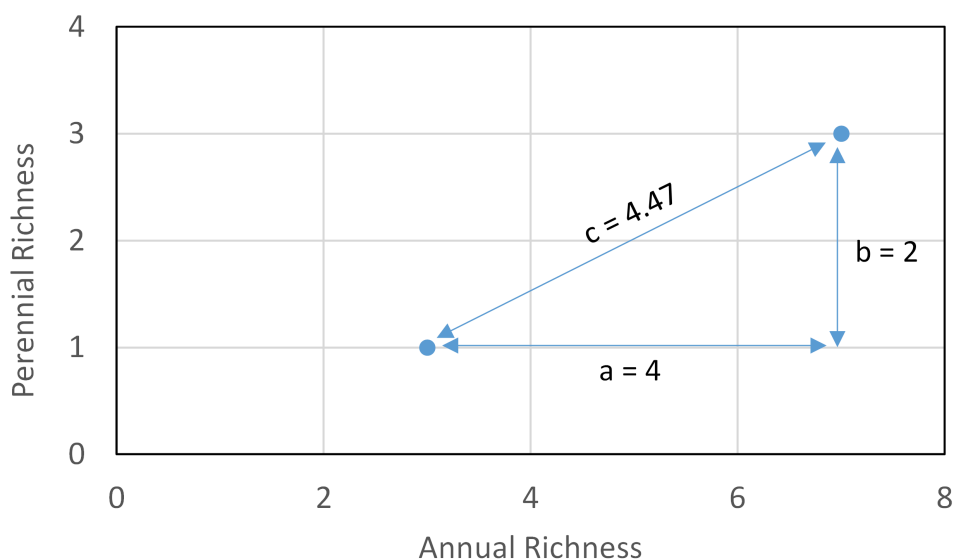
Plot	Annual Richness	Perennial Richness
H	3	1
I	7	3

Total richness is the same as before, but what is the distance between the two plots now?

A reasonable first step is to use the Pythagorean theorem ($a^2 + b^2 = c^2$) to calculate the Euclidean distance (ED) between the two plots:

$$ED = \sqrt{(x_{HA} - x_{IA})^2 + (x_{HP} - x_{IP})^2} = \sqrt{(3 - 7)^2 + (1 - 3)^2} = 4.47 \text{ species}$$

Incorporating information about the longevity of the plant species has reduced the distance between the plots from 6 species to 4.5 species. A visualization of this calculation is shown below.



Distance between two sample units in terms of the richness (number of species) of annual and perennials, as calculated with the Pythagorean theorem.

Desirable Properties of Distance Measures

We will consider **five desirable properties** of distance measures (more have been suggested – see Legendre & De Cáceres (2013) for details).

1) Zero If Identical

If sample units A and B have the same values for all variables, the distance between them should be **zero**.

This is true of all distance measures we will consider.

2) Positive

If sample units A and B do not have the same values for all variables (i.e., are not identical), the distance between them should be **positive**. Since the distance is zero when they are identical (property 1), what would a negative distance mean?

This is true of all distance measures we will consider.

3) Symmetric

A distance measure is **symmetric** if the distance from A to B equals the distance from B to A.

This is true of all distance measures we will consider. When would a distance measure not be symmetric? Some examples are found in mapping applications. In essence, these applications seek the shortest distance between two points. However, there are multiple scenarios in which the shortest distance between two points is not the same:

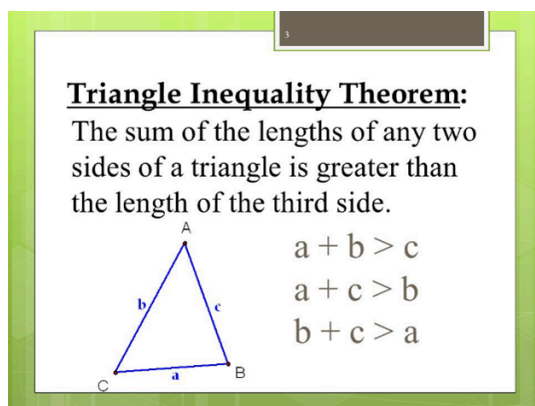
- Driving directions, especially when there are one-way streets. The roads you use to get to a location are not the same ones you would use to drive back from that location. This is a special form of the Manhattan or city block distance. More information: https://en.wikipedia.org/wiki/Taxicab_geometry.
- Estimates of travel time between destinations that incorporate elevation and account for the fact that it is more work to walk uphill than downhill. For example, Google maps estimates that it would take 4 minutes less time for me to walk to UW than home from it, probably due both to slightly different route recommendations and to the elevation difference. This example is modified from [https://en.wikipedia.org/wiki/Metric_\(mathematics\)#Quasimetrics](https://en.wikipedia.org/wiki/Metric_(mathematics)#Quasimetrics).
- As an ecological example, Acevedo et al. (2015) studied a wind-dispersed orchid and showed that models which accounted for wind direction more accurately predicted its colonization and extinction dynamics. In other words, if sample unit A is downwind of sample unit B, the effective distance between them was smaller from B to A than from A to B – pollen would have to travel ‘farther’ upwind from A to B.

4) Metric or Semimetric?

When we have multiple sample units, we can calculate distances between many pairs of sample units. For example, if we have three sample units (A, B, C), we can calculate the distance from A to B, A to C, and B to C. Imagine the distances between A, B, and C as the sides of a triangle with the vertices representing the sample units.

A **metric** distance measure follows the principles of Euclidean geometry in what is known as the **triangle inequality theorem** (see image below): the distance from one sample unit to another (e.g., A to C) is always smaller than the combined distance from the first to the other by way of a third (e.g.,

A to B and B to C). Measures that satisfy this inequality are most properly referred to as 'distance measures'.



Measures that do not satisfy the triangle inequality theorem are **semimetric**. With these measures, Euclidean geometry may not work – the distance from A to C can be greater than the combined distance from A to B and B to C. These are properly referred to as '**dissimilarity measures**'. See the Sorenson dissimilarity in the next chapter for an example.

5) Is There a Constant Maximum?

Some measures do not have an upper limit – no matter what the distance is between two sample units, you can conceive of a situation in which the distance would be greater. For example, consider the physical distance between sample units. No matter how far apart two sample units are, you can envision another pair of sample units that are further apart.

Other measures have a **constant maximum**, meaning that there is no situation in which the distance could be greater. This can happen when samples have no elements in common, and often arises when the distance is expressed as a proportion of some total – doing so bounds the values to be less than or equal to 1. Many of these are dissimilarity measures.

The presence of a constant maximum permits a dissimilarity measure to also be expressed as a similarity measure. For example, if two sample units have a dissimilarity of 0.25 in a measure with a maximum dissimilarity of 1, it would be equivalent to say that they have a similarity of 0.75 (i.e., $1 - 0.25$).

The Distance Matrix

When a distance measure is applied to multiple plots, a distance is calculated for every pairwise combination of plots. These distances are then assembled into a **distance matrix** (or dissimilarity matrix). For example, here is a distance matrix between three plots (A, B, C):

	A	B	C
A	0.0	0.8	0.4
B	0.8	0.0	0.5
C	0.4	0.5	0.0

This matrix has several important features:

- It is **square** – recall from the matrix algebra chapter that many of the manipulations possible with matrix algebra are applied to square matrices.
- It is **symmetric** (desirable property #3) – for example, the distance from A to B is the same as the distance from B to A. This means that the upper-triangle is a mirror image of the lower-triangle.
- The distance between a plot and itself is zero (desirable property #1) – all values along the **diagonal** are zero.
- The first row and last column are non-informative – they contain information that is also reported elsewhere in the matrix.

As a result of these features, we often display a distance matrix more concisely as a **lower triangular matrix**:

	A	B
B	0.8	
C	0.4	0.5

Although this doesn't look like a matrix and is neither square or symmetric, it is still described as a distance matrix. Furthermore, even though it may appear to have only two rows and two columns it is still a 3 x 3 distance matrix.

A distance matrix summarizes the distances between every pair of sample units.

How does the number of pairwise distances scale with the number of sample units? There are $n \times n = n^2$ pairwise combinations of n sample units. However, since a distance matrix is symmetric with zeroes on the diagonal, **the number of unique pairwise combinations is**

$$\frac{n(n-1)}{2}.$$

How many unique pairwise combinations are there for:

3 sample units? _____

10 sample units? _____

20 sample units? _____

The Distance Matrix

The distance matrix is square and symmetric, with zeroes on the diagonal. Therefore, it is often shown simply as its lower triangle.

The number of pairwise distances in the matrix is a function of the number of sample units.

The number of variables has no effect on the size of the distance matrix.

Conclusions

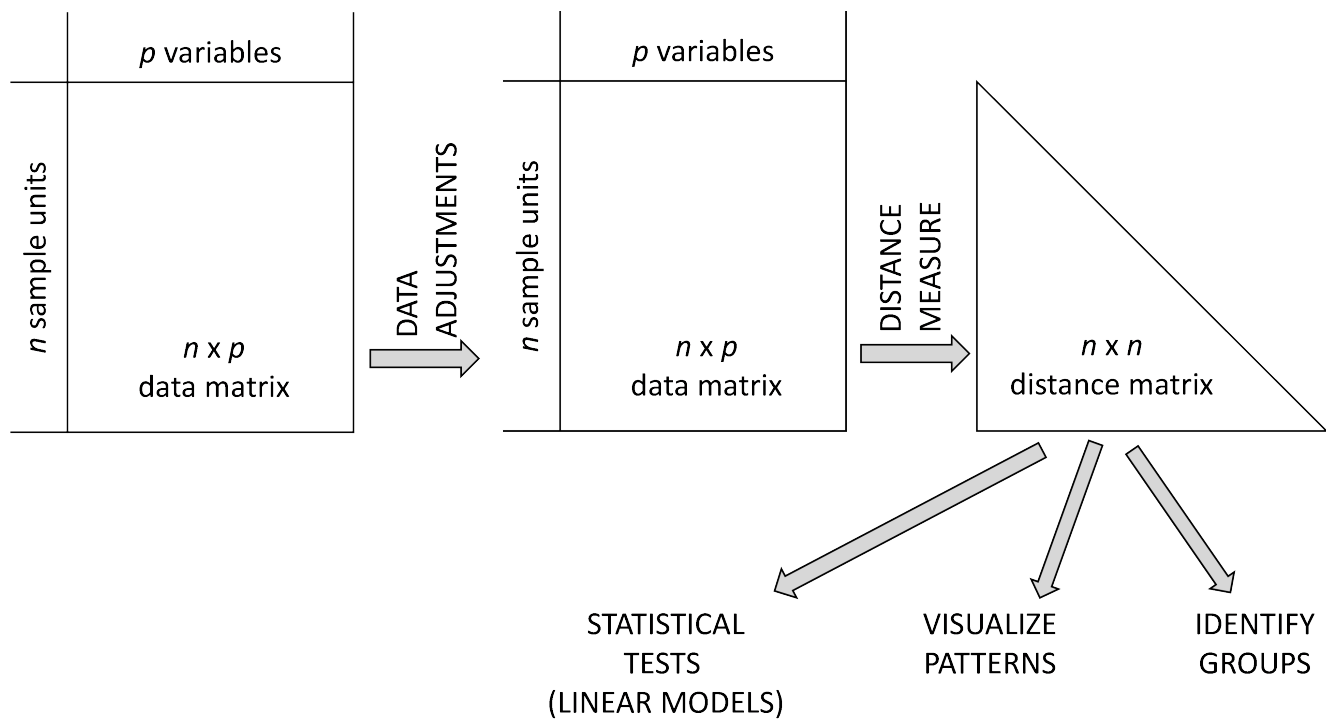
The calculation of a distance between two sample units compresses or combines all of the differences between two samples into a single number. This 'compression' occurs regardless of whether the samples are being compared with respect to a single variable (e.g., species richness) or a multivariate measure (e.g., community composition). Can you see how this is so?

Another way of stating this is that the size of the distance matrix is a function of the number of sample units, not the number of variables. Because a distance matrix is unaffected by the number of variables, **distance-based techniques can be applied identically to both univariate and multivariate data**; separate techniques are not required as is the case with conventional parametric techniques (e.g., ANOVA vs. MANOVA).

There are many ways to combine the differences between sample units. The next chapter reviews a number of these distance measures.

The rest of this course builds upon the foundation laid up to this point. Once we have made all desired data adjustments and expressed the differences among samples in a distance matrix, we can use that distance matrix to test for differences among pre-existing groups, visualize patterns, identify natural groups in the data, etc.

This road map might clarify the process:



References

Acevedo, M.A., R.J. Fletcher Jr., R.L. Tremblay, and E.J. Meléndez-Ackerman. 2015. Spatial asymmetries in connectivity influence colonization-extinction dynamics. *Oecologia* 179(2):415-424.

Legendre, P., and M. De Cáceres. 2013. Beta diversity as the variance of community data: dissimilarity coefficients and partitioning. *Ecology Letters* 16:951-963.

Media Attributions

- richness
- triangle.inequality
- analysis.road.map

12. Common Distance Measures

Learning Objectives

To consider a range of distance measures used with ecological data, and the types of data for which they are appropriate.

To consider whether shared absences matter and how to deal with empty sample units.

To continue using R.

Resources

Legendre & De Cáceres (2013)

Key Packages

```
require(tidyverse, vegan, labdsv, ecodist, betapart)
```

Introduction

Distance measures are an essential component of many ecological analyses. There are many to choose from; Legendre & De Cáceres (2013) compared 16 of them, and Legendre & Legendre (2012, Table 7.2) list 26 of them! Borcard et al. (2018) devote an entire chapter to distance measures, using two primary criteria to organize their discussion:

- What are the **distributional characteristics** of the data? For example, are the data binary (e.g., presence/absence) or continuously distributed (e.g., abundance)?
- Are **shared absences** meaningful? A shared absence is a species (or other variable) that is absent from both sample units under consideration. Data are 'symmetric' if a shared absence is meaningful, and 'asymmetric' if it is not. Symmetry in this case means that two samples having the same zero value is as meaningful as those samples having another value the same (e.g., if the value of a variable was 1.5 for both samples). Species composition data are a prime example of a situation where shared absences are not meaningful, as discussed in the 'Two

Issues to Consider' section below.

Combinations of these criteria are most appropriately handled using different types of distance measures. A few distance measures are identified here; those discussed below are in bold.

	Species data (asymmetric)	Non-species data (symmetric)
Quantitative (continuous)	Bray-Curtis Chi-Square UniFrac	Euclidean Manhattan
Mixed, including categorical		Gower
Binary	Jaccard Sorenson UniFrac	Simple matching

Note: the simple matching distance measure is not commonly used in ecology, and is not discussed here.

Euclidean Distance

The Pythagorean theorem is easily visualized in two dimensions, as we did in the last chapter. It can also be applied to more dimensions, though it rapidly becomes difficult to visualize this.

The formula for the Euclidean Distance (*ED*) between samples *i* and *h* across *p* dimensions is:

$$ED = \sqrt{\sum_{j=1}^p (a_{hj} - a_{ij})^2}$$

Here is a dataset reporting the presence or absence of each of five species (variables) on three plots:

Plot	SppA	SppB	SppC	SppD	SppE
1	1	1	1	0	0
2	0	0	0	1	1
3	1	1	1	1	1

(Source: Legendre & Legendre 2012, p. 311)

What is the Euclidean distance (ED) between each pair of plots?

ED(1,2) = _____

ED(1,3) = _____

ED(2,3) = _____

Verify that the distance from a plot to itself is zero (property 1), and that the distance from plot 1 to plot 2 is the same as the distance from plot 2 to plot 1 (property 3). Euclidean distances can take any non-negative value from 0 to infinity (property 2).

Euclidean distances can be calculated using positive and negative values. One potential limitation of this distance measure is that the **calculated distance depends on the scale of the variables**. For example, if variables are measured using very different scales (e.g., biomass in g for forbs, Mg for trees), the distances will be disproportionately affected by the variables measured using the larger scales (Legendre & Legendre 2012). However, this can be addressed by relativizing the variables appropriately before calculating distances.

Euclidean distances are appropriate for many types of data, including geographic distances. However, **Euclidean distances are generally inappropriate for community data** (e.g., a plot x species matrix containing the cover or presence/absence of multiple species). Why? One reason is that it's possible for two samples with no species in common to have a smaller Euclidean distance than two samples that share species. For example, compare the Euclidean distances among the following plots:

Plot	SppA	SppB	SppC
4	0	4	8
5	0	1	1
6	1	0	0

(Source: modified from Legendre & Legendre 2012, Figure 7.8)

ED(4,5) = _____

ED(4,6) = _____

ED(5,6) = _____

Verify that plots 5 and 6 are more similar than plots 4 and 5. Many ecologists find this unsatisfying because plots 5 and 6 have no species in common whereas plots 4 and 5 share the same species and differ only in abundance. Instead, they would argue that the presence of the same species is more important than a difference in abundance of that species.

Manhattan Distance

Euclidean distances are calculated by squaring the difference associated with each variable, but an alternative is to simply add the (absolute) differences. This is analogous to summing the two perpendicular sides of a triangle rather than using the Pythagorean theorem to calculate the hypotenuse.

The formula for the Manhattan distance between samples i and h across p dimensions is:

$$MD = \sum_{j=1}^p | (a_{hj} - a_{ij}) |$$

In the last chapter, we used the Euclidean distance to calculate the hypotenuse between two sample

units based on the numbers of annual and perennial species. The Manhattan distance between these sample units is the sum of the two perpendicular sides of the triangle.

This is also called the city-block distance ... can you see why?

Jaccard Similarity and Dissimilarity

Let's consider presence/absence data some more. For any two plots, species occurrences can be summarized in a contingency table:

		Plot B	
		Present	Absent
Plot A	Present	a	b
	Absent	c	d

Note that this is **not** a data matrix. Rather:

- a is the number of species that are present in both plots
- b is the number of species that are present in plot A but missing from plot B
- c is the number of species that are missing from plot A but present in plot B
- d is the number of species that are missing from both plots.

Jaccard (1912) proposed that we quantify the proportion of species that are present in both samples. This is known as Jaccard similarity (S_J):

$$S_J = \frac{a}{a + b + c}$$

As a proportion, this value is bounded between 0 (no shared species) and 1 (all shared species). This is a metric measure.

Note that species that are missing from both plots (d) are not included in this calculation; see the 'Two Issues to Consider' section below for more information on this.

Since Jaccard similarity has an upper bound of 1, it is converted to Jaccard dissimilarity (D_J) by subtraction. D_J can also be calculated directly from the contingency table of species occurrences:

$$D_J = 1 - S_J = \frac{b + c}{a + b + c}$$

Jaccard dissimilarity is the proportion of species that are absent from one of the samples.

Refer back to plots 1-3 for which we calculated Euclidean distances. What is the Jaccard dissimilarity between each pair of plots?

	Plot1	Plot2
Plot2		
Plot3		

Note: Recent work has decomposed or partitioned Jaccard dissimilarities into two components, turnover and nestedness (Baselga 2010, 2012). Turnover is species replacement (one species replaced with another), while nestedness is the extent to which the composition of one sample unit is a subset of the composition of another sample unit. See the description of the `betapart` package below for more information.

Sorensen Similarity and Dissimilarity

Sorensen similarity (S_S) is the proportion of species that are present in both samples, while accounting for differences in species richness between samples. Using the same terminology as for Jaccard similarity, the formula is:

$$S_S = \frac{a}{\frac{(a+b)+(a+c)}{2}} = \frac{2a}{2a + b + c}$$

Like Jaccard similarity, this value is bounded between 0 (no shared species) and 1 (all shared species). Unlike Jaccard, however, it is semimetric.

Several people proposed this distance measure independently; the original publication by Sørensen is from 1948.

Since S_S is a proportion, it can be converted to Sorensen dissimilarity (D_S) by subtraction. D_S can also be calculated directly from the contingency table:

$$D_S = 1 - S_S = 1 - \frac{2a}{2a + b + c} = \frac{b + c}{2a + b + c}$$

Sorensen dissimilarity is the proportion of species that are absent from one of the samples.

Refer back to plots 1-3. What is the Sorensen dissimilarity between each pair of plots?

	Plot1	Plot2
Plot2		
Plot3		

Notice that these data do not satisfy the triangle inequality: the dissimilarity from plot 1 to 3 plus the dissimilarity from plot 3 to plot 2 is less than the dissimilarity from plot 1 to 2. This demonstrates that the Sorensen dissimilarity is a semimetric measure.

Bray-Curtis Distance

When the formula for Sorensen dissimilarity is extended from presence/absence data to species abundance data, it results in the Bray-Curtis distance measure:

$$D_{i,h} = \frac{\sum_{j=1}^p |a_{ij} - a_{hj}|}{\sum_{j=1}^p a_{ij} + \sum_{j=1}^p a_{hj}} = 1 - \frac{2 \sum_{j=1}^p \text{MIN}(a_{ij}, a_{hj})}{\sum_{j=1}^p a_{ij} + \sum_{j=1}^p a_{hj}} = 1 - \frac{2 \sum_{j=1}^p \text{MIN}(a_{ij}, a_{hj})}{a_{i\cdot} + a_{h\cdot}}$$

where

- p is the total number of species
- a_{ij} is the abundance of species j in sample unit i
- a_{hj} is the abundance of species j in sample unit h
- $a_{i\cdot}$ is the total abundance of all species in sample unit i
- $a_{h\cdot}$ is the total abundance of all species in sample unit h

These formulae are from chapter 6 of McCune & Grace (2002). The middle and right-hand versions are the same except that in the right-hand one I used the same terminology in the denominator as in the formula for the chi-square distance below. This is to permit easier comparisons between the two measures. Note that these formulae are based on the data matrix, not on the contingency table that was the basis of the Sorensen dissimilarity.

The Bray-Curtis distance measure is bounded between 0 (the sample units are identical) and 1 (the sample units are completely different), and is semimetric.

The Bray-Curtis distance measure is named after the co-authors of the paper in which it was used (Bray & Curtis 1957). However, and confusingly, it is also known by many other names: **Steinhaus**, **Czekanowski**, **Sorensen**, and **percentage difference**. Often this is because the same measure was proposed independently or because two measures were proposed that were later shown to be mathematically equivalent.

Several studies (notably, Faith et al 1987) have concluded that **the Bray-Curtis distance measure functions best for community data** (e.g., a plot x species matrix). We will see it throughout this course.

Borcard et al. (2018) note that the Bray-Curtis distance “gives the same importance to absolute differences in abundance irrespective of the order of magnitude of the abundances ... a difference of 5 individuals has the same weight when the abundances are 3 and 8 as when the abundances are 6203 and 6208” (p.39). If this is problematic, the data can be log-transformed before computing distances.

Note: Recent work has decomposed or partitioned Bray-Curtis distances into two components, one related to ‘balanced variation in abundance’ and the other to ‘abundance gradients’ (Baselga 2013). These components are analogous to the turnover and nestedness components of Jaccard dissimilarities. See the description of the `betapart` package below for more information.

Chi-Square Distance

The Chi-square distance measure is the basis of an ordination technique known as **correspondence analysis**, which, with its variants, is popular in some quarters. Simulation tests have found that chi-

square distances do not perform well with community data (Faith et al. 1987), but the popularity of correspondence analysis means that it is helpful to be familiar with this measure.

The formula for chi-square distance is:

$$D_{i,h} = \sqrt{\sum_{j=1}^p \frac{1}{a_{.j}} \left[\frac{a_{hj}}{a_{h.}} - \frac{a_{ij}}{a_{i.}} \right]^2}$$

where

- p is the total number of species
- a_{ij} is the abundance of species j in sample unit i
- a_{hj} is the abundance of species j in sample unit h
- $a_{i.}$ is the total abundance of all species in sample unit i
- $a_{h.}$ is the total abundance of all species in sample unit h
- $a_{.j}$ is the total abundance of species j across all sample units

This formula is from chapter 6 of McCune & Grace (2002).

Like Euclidean distances, this measure involves summing squared differences. However, the chi-square distance measure also:

- Expresses the abundance of each species (a_{hj} and a_{ij}) as a proportion of the total abundance on the sample unit ($a_{h.}$ and $a_{i.}$). In other words, relativization by row total is built into this distance measure.
- Weights the squared difference by the inverse of the total abundance of the species ($a_{.j}$). This is also a built-in relativization, but it's not one of the common ones that we've discussed previously (e.g., relativization by column maximum). This is relativization by column total.

This last aspect – weighting by the total abundance of the species – is where some of the problems arise. It means, for example, that:

- The distance between two sample units depends on which other sample units are included in the data matrix.
- Since this is an inverse weighting, common species are downplayed and rare species are weighted more strongly.

UniFrac Distance

In the above distances, each species is considered individually and the distance between them are summed to determine the total distance between sample units. Another approach is to account for the degree of relatedness among species. For example, consider these three plots (for simplicity, I only show three species):

Plot	<i>Poa pratensis</i>	<i>Poa compressa</i>	<i>Hypochaeris radicata</i>
7	5	0	0
8	0	5	0
9	0	0	5

By any of the above distance measures, the magnitude of the difference is the same between any two plots. However, the species in plots 7 and 8 are from the same genus and thus arguably these plots are more similar to one another than to plot 9.

UniFrac incorporates phylogenetic information about the taxa in the distance between sample units. The taxa present in two sample units are placed on a phylogenetic tree. Every branch in this tree ends at a taxon, and the length of the branch is a measure of how related that taxon is to the next one to which it connects. A branch is coded as 'shared' if the taxon is present in both sample units and as 'unshared' if the taxon is only present in one sample unit. I haven't shown the distance formula here, but it is simply the sum of unshared branch lengths as a proportion of the total of all tree lengths. This is a metric measure with values ranging from 0 to 1.

The original UniFrac measure (Lozupone & Knight 2005) is unweighted, meaning that it gives equal weight to each taxa (analogous to Jaccard and Sorenson dissimilarities). There is also a weighted UniFrac which accounts for the relative abundances of taxa (analogous to Bray-Curtis distance) as well as a generalized UniFrac that combines the weighted and unweighted approaches in a single framework (Chen et al. 2012). More recently, this approach has been adapted for use in paired and longitudinal designs as are commonly used in microbiome studies (Plantinga et al. 2019).

This distance measure is not as easy to apply as the others as it requires phylogenetic information in addition to the standard composition matrix (sample units x species). It obviously is only relevant for compositional data.

Gower's Distance

Most distance measures assume that the underlying data are continuously distributed, but Gower (1971) proposed a generalized coefficient of dissimilarity that can be applied to a dataset consisting of a variety of data types: continuously distributed, nominal, and/or ordinal variables. Greenacre & Primicerio (2013) provide a nice description of this approach. Gower's distance is available in several of the functions summarized below.

Mahalanobis Distance

Mahalanobis (1936) proposed a way to compare a sample to sets of other samples and determine how likely the sample belongs to each set. The formula includes the covariance matrix to account for differences in variability among variables. It is intended for multivariate normal data.

One way the Mahalanobis distance can be used is to evaluate whether individual sample units are outliers relative to the rest of the data – see the 'Multivariate Outlier Analysis' chapter for more information.

Should Shared Absences Matter?

When summarizing species data, ecologists generally prefer distance measures that are not affected by species that are absent from the two experimental units being compared (cell *d* in the contingency table shown in the description of Jaccard similarity above). Species could be absent for any number of reasons, and it generally doesn't make sense to determine the similarity between two samples based on species that are present in neither.

However, some research has shown that species absences can be informative when dealing with datasets that contain high beta diversity (species turnover among plots). For more information about 'extended dissimilarities', see De'ath (1999) and Boyce & Ellison (2001), the discussion of the `dsydis()` function below, and the help files for the `stepacross()` function in `vegan`. Note that using a function such as `stepacross()` will change how distances are calculated such that they no longer have a fixed upper bound. Anderson et al. (2011) provide an extended discussion about beta diversity.

Dealing With Empty Sample Units

We've mentioned before that most distance measures assume that at least one species is present in all plots. This can be problematic if you are dealing with a community in which organisms are heterogeneously distributed – their absence from a given area may be very important information! For example, consider these data:

Plot	SppA	SppB	SppC
4	0	4	8
5	0	1	1
6	1	0	0
10	0	0	0

These are plots 4-6 from above, together with plot 10 which is empty – there were no species in this sample unit. There are two common ways to deal with this.

First, you could delete the empty sample units, as we discussed earlier (see 'Data Adjustment' chapter). However, this changes the questions being asked – for example, from “how does composition differ among all plots?” to “how does composition differ among vegetated plots?”.

Second, you could use a 'zero-adjusted' distance measure (Clarke et al. 2006). This simply involves adding a 'pseudo-species' with the same small cover value to all plots. Often, the cover value used is the minimum value that a species would be assigned if present on a plot. This pseudo-species is added to all plots – not just those that are empty – so that it alters all distances in the same manner. This approach can be applied to data subject to any distance measure. You would want to exclude pseudo-species when calculating metrics such as species richness.

Using one of the functions described below, verify that you cannot calculate the Bray-Curtis distance between plot 10 and any of the other plots. Then, add a 'dummy' species with an abundance of 1 in all plots and re-calculate the distance matrix.

Some code that may be of interest:

```
eg <- data.frame(
  Plot = c("p4", "p5", "p6", "p10"),
  SppA = c(0, 0, 1, 0),
```

```
SppB = c(4, 1, 0, 0),
SppC = c(8, 1, 0, 0) )
Spp <- c("SppA", "SppB", "SppC")
vegan::vegdist(eg[ 1:3 , colnames(eg) %in% c(Spp)])
eg$SppDummy <- 1
vegan::vegdist(eg[ , colnames(eg) %in% c(Spp, "SppDummy")])
```

R Functions to Calculate Distance Measures

The above distance measures, and others, can be calculated using many R functions, including:

- `dist()` in `stats`
- `vegdist()` and `betadiver()` in `vegan` (this package also includes the `designdist()` function for constructing your own distance measure)
- `dsvdis()` in `labdsv`
- `distance()` and `bcdist()` in `ecodist`
- `gdist()` and `xdiss()` in `mvpart`
- `daisy()` in `cluster`
- `gowdis()` in `FD`
- `beta.pair()` and `bray.part()` in `betapart`

We'll survey a few of these functions here. Don't forget to load these packages before calling their functions! Details about the measures calculated by each function can be obtained through the R help files.

Keep in mind that many distance measures are referred to by several equivalent names.

stats::dist()

The `dist()` function is part of the `stats` package which is part of the base R installation and thus is automatically loaded. It can calculate 6 different distance measures. Its usage is:
`dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)`

The key arguments are:

- `x` – the data matrix, data frame, or distance matrix to be analyzed
- `method` – the distance measure to be used. There are 6 distance measures available:
 - `euclidean` – the default
 - `maximum` – the largest absolute difference among all pairwise elements.
 - `manhattan` – aka City-block. Calculated as the sum of the absolute differences along all dimensions.
 - `canberra` – Sum across all pairwise elements of the absolute value of the difference divided by the absolute value of the sum.
 - `binary` – proportion of pairwise elements where one of the two is non-zero. Pairs where both are zero are ignored.
 - `minkowski` – generalized form of Euclidean and Manhattan distances (see p. 47 of McCune & Grace (2002) for equation). Requires an additional argument specifying the square root and power to be applied; the default is `p = 2`.
- `diag` – print the diagonal of the distance matrix? Default is `FALSE` (no).

- `upper` – print the upper-triangle of the matrix? Default is `FALSE` (no).
- `p` – power; used only when calculating the Minkowski distance.

`vegan::vegdist()`

The `vegdist()` function in `vegan` currently includes 21 distance measures. Its usage is:

```
vegdist(x, method = "bray", binary = FALSE, diag = FALSE, upper = FALSE, na.rm = FALSE, ...)
```

The key arguments are:

- `x` – the data matrix to be analyzed, with plots as rows and variables as columns
- `method` – type of distance measure to use. Notes about a few of the distance measures follow; the others can be found in the R help files.
 - `manhattan` – identical to result from `dist()`
 - `euclidean` – identical to result from `dist()`, but not the default measure.
 - `canberra` – may yield different results than from `dist()`
 - `clark`
 - `bray` – Bray-Curtis. The default measure for this function.
 - `kulczynski`
 - `jaccard` – computed as $2B/(1+B)$, where B is the Bray-Curtis dissimilarity. The help file indicates that Bray-Curtis and Jaccard dissimilarities are rank-order similar and suggests that the Jaccard “probably should be preferred” because it is metric whereas Bray-Curtis is semimetric. However, I have not seen any formal evaluation of how sensitive conclusions are to this choice.
 - `gower` – the help file indicates that this version of Gower’s distance cannot handle mixed data (e.g., continuous variables and factors simultaneously). I have not tested this.
 - `altGower`
 - `morisita`
 - `horn`
 - `mountford`
 - `raup`
 - `binomial`
 - `chao` – a measure that accounts for unseen species pairs (i.e., differences in sample size that particularly affect rare species). See Chao et al. (2005) for details.
 - `cao`
 - `mahalanobis`
 - `chisq` – Chi-square distances, as used in correspondence analysis.
 - `chord`
 - `aitchison`
 - `robust.aitchison`
- `binary` – whether to use `decostand()` to convert data to presence/absence before calculating distances. Default is `FALSE` (no).
- `diag` – whether to return the diagonal of the distance matrix. Default is `FALSE` (no).
- `upper` – whether to return the values in the upper triangle of the distance matrix. Default is `FALSE` (no).
- `na.rm` – whether to delete missing observations (pairwise) when calculating dissimilarities. Default is `FALSE` (no).

labdsv::dsvdis()

The `dsvdis()` function in `labdsv` currently offers 7 distance measures. Its usage is:

```
dsvdis(x, index, weight = rep(1, ncol(x)), step = 0.0, diag = FALSE, upper = FALSE)
```

The key arguments are:

- **x** – the data matrix to be analyzed, with plots as rows and variables as columns
- **index** – the distance measure to be used. Note that this argument has a different name than the comparable argument in the other functions ('index' vs 'method') and does not have a default distance measure. The available distance measures are:
 - **steinhaus** – aka Jaccard. Based on presence/absence (converts abundances automatically).
 - **sorensen** – based on presence/absence (converts abundances automatically).
 - **ochiai** – based on presence/absence (converts abundances automatically).
 - **ruzicka**
 - **bray/curtis** – Bray-Curtis. Note the diagonal in the name here.
 - **roberts**
 - **chisq**
- **weight** – an opportunity to weight species differently during the calculation of distances. The default (`rep(1, ncol(x))`) assigns all species the same value (1).
- **step** – a threshold dissimilarity to initiate shortest-path adjustment. This is likely to be most useful in datasets where there is complete species turnover (e.g., spanning very large gradients). The default (0.0) means that no adjustments are made. If a positive value is specified, any distances above that value are replaced by the distance of the shortest connected path between points less than the threshold apart. The result is that the dissimilarity values are no longer bounded between 0 and 1. To my knowledge, the consequences of these types of adjustments for ecological interpretation have not been rigorously assessed.
- **diag** – whether to return data for the diagonal. Default is **FALSE** (no).
- **upper** – whether to return data in the upper triangle of the distance matrix (**TRUE**) or the lower triangle (**FALSE**). Default is **FALSE** (no).

ecodist::distance()

For an introduction to the `ecodist` package, see Goslee & Urban (2007). In `ecodist`, the `distance()` function can calculate 10 different distance measures. Its usage is:

```
distance(x, method = "euclidean", sprange = NULL, spweight = NULL, icov, inverted = FALSE)
```

The key arguments are:

- **x** – the data matrix or data frame to be analyzed, with plots as rows and variables as columns
- **index** – the distance measure to be used. There are 10 distance measures available:
 - **euclidean** – the default method for this function.
 - **bray-curtis** – note the dash in the name here. Can also be calculated directly using the `bcdist()` function in this package.
 - **manhattan**

- `mahalanobis`
 - `jaccard`
 - `difference`
 - `sorensen`
 - `gower` – Gower's distance.
 - `modgower10` – variation of Gower's distance measure, using base 10.
 - `modgower2` – variation of Gower's distance measure, using base 2.
- `sprange` and `spweight` permit species data to be relativized during the distance calculation. These only apply to a subset of the available distance measures, particularly Gower's distance and variations on it.
 - `icov` and `inverted` are used when calculating Mahalanobis distances.

This package also includes `bcdist()`, a function that returns the Bray-Curtis distance measure. This is faster than `distance(x, index = "bray-curtis")`, and includes an option to drop empty rows or to set distances between empty rows to zero.

betapart

The `betapart` package provides functions to partition dissimilarity measures into their components.

Incidence-based metrics can be partitioned into the variation associated with turnover and with nestedness. The `beta.pair()` function calculates three distance matrices, one for turnover, one for nestedness, and one for total dissimilarity. It can be used to calculate either Jaccard or Sorensen dissimilarities.

The components of abundance-based metrics are similar to those of the incidence-based metrics but a bit nuanced: balanced variation in abundance instead of turnover, and abundance gradients instead of nestedness. The `bray.part()` function calculates three distance matrices, one for balanced variation, one for abundance gradients, and one for total Bray-Curtis dissimilarity.

The components of dissimilarity are additive – the sum of the first two distance matrices calculated using either method above is equal to the third distance matrix (total dissimilarity). The components can be analyzed separately but, since they are aspects of the same value, I often relativize them and consider the proportion of the total dissimilarity that is attributable to one of the two components.

Examples

Today's Example Dataset

The dataset of plots 1-3 used to introduce the Euclidean distance measure is available as a text file (`Legendre.Legendre.2012.p311.txt`) in the course GitHub folder. Save it in the 'data' subfolder within your course folder. Then, open your R project and load the data into R:

```
test <- read.table("data/Legendre.Legendre.2012.p311.txt", header = TRUE)
```

To see the Euclidean distances among plots in `test`:

```
(ED.test <- dist(test))
```

Note that the result is displayed as a lower triangular matrix. What class is this object? It can easily be converted to a full matrix:

```
as.matrix(ED.test)
```

Oak Plant Community Dataset

Now, let's calculate a distance matrix using the geographic data associated with our Oak plant community dataset. Assuming you have already saved the files to your data folder, we begin by loading them:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
```

Create an object containing the response data:

```
Oak_abund <- Oak[, colnames(Oak) %in% Oak_species$SpeciesCode]
```

As we've already seen, this dataset includes a number of potential explanatory variables along with the abundances of a large number of species (see the 'Oak_Metadata.docx' file for details). We will begin by focusing on the latitude and longitude of each stand. We will extract these variables and then use them to calculate a distance matrix. The Euclidean distance measure is a reasonable choice since these data are spatial coordinates.

```
library(vegan)
geog.dis <- Oak[,c("LatAppx", "LongAppx")] |>
vegdist(method = "euc")
```

The resulting object contains 1081 distances:

```
length(geog.dis)
```

Why this many distances? Recall from the last chapter the formula for the number of pairwise distances among n sample units.

Note that the number of variables has no effect on the size of the resulting distance matrix. We will illustrate this by also calculating a distance matrix based on the species abundance data. Before doing so, let's think about relativizations again. In this dataset, trees were measured in very different units than other growth forms so it makes sense to relativize each species by its maximum – this will put the data for all species on the same scale. (If appropriate for our objectives, we could also have made other adjustments, such as deleting rare species and relativizing by row totals. For simplicity, we did not do so here.) We can do this within our piped functions:

```
spp.dis <- Oak_abund |>
  decostand(method = "max") |>
  vegdist()
```

I didn't specify a method for `vegdist()`. Why not?

Use the `str()` function to view the details of `geog.dis` and `spp.dis`. Verify that they are the same size, even though `geog.dis` was based on two variables and `spp.dis` was based on 189 variables.

Distance matrices like these are what we are going to utilize throughout the rest of the course.

Conclusions

The behavior of a distance measure is highly influenced by the data adjustments (deletions, transformations, standardizations/relativizations) applied prior to its calculation. It bears repeating that the appropriateness of these various adjustments must be determined based on the questions you are addressing.

A single distance matrix always contains values calculated using the same distance measure – we would never combine in one distance matrix measures obtained using different distance measures. However, we will see examples of how matrices derived from different data on the same sample units (as with `geog.dis` and `spp.dis` above) can be compared.

There are many distance measures to choose from. While this can be confusing, it also gives flexibility. Distance measures have different properties and/or make different assumptions about the data. These properties and assumptions are important to keep in mind when choosing which distance measure to use. Some analytical techniques implicitly assume a distance measure:

- Correspondence Analysis (CA) and Canonical Correspondence Analysis (CCA) are based on chi-square distances
- Principal Component Analysis (PCA) are based on Euclidean distances

In instances like these, any limitations of the assumed distance measure are carried over to the associated technique. For example, if it is inappropriate to summarize a dataset using Euclidean distances, it would also be inappropriate to analyze that dataset using PCA. Conversely, if Euclidean distances are appropriate to apply to a dataset, then PCA may be appropriate ... depending on other considerations such as the degree of correlation. See the 'PCA' chapter for additional information.

The fact that distance measures have different properties can be an asset. For example, a compositional matrix could be summarized with a presence/absence-based measure and with an abundance-based measure. Rare species are given more weight in a presence/absence measure whereas common species are given more weight in an abundance-based measure. Therefore, if these two distance matrices were analyzed identically, differences between the resulting analyses would reflect whether patterns in the community are being driven by the rare or the common species (Lozupone et al. 2007). In a recent study (Bakker et al. 2023) I compared abundance- and incidence-based measures of grassland compositional variation. We found that grasslands differed greatly in the importance of these two measures, and that these measures were correlated with different aspects of the environmental conditions at a site.

References

Anderson, M.J., T.O. Crist, J.M. Chase, M. Vellend, B.D. Inouye, A.L. Freestone, N.J. Sanders, H.V. Cornell, L.S. Comita, K.F. Davies, S.P. Harrison, N.J.B. Kraft, J.C. Stegen, and N.G. Swenson. 2011. Navigating the multiple meanings of β diversity: a roadmap for the practicing ecologist. *Ecology Letters* 14:19-28.

Bakker, J.D., J.N. Price, J.A. Henning, E.E. Batzer, T.J. Ohlert, C.E. Wainwright, P.B. Adler, J. Alberti, C.A. Arnillas, L.A. Biederman, E.T. Borer, L.A. Brudvig, Y.M. Buckley, M.N. Bugalho, M.W. Cadotte, M.C. Caldeira, J.A. Catford, Q. Chen, M.J. Crawley, P. Daleo, C.R. Dickman, I. Donohue, M.E. DuPre, A. Ebeling, N. Eisenhauer, P.A. Fay, D.S. Gruner, S. Haider, Y. Hautier, A. Jentsch, K. Kirkman, J.M.H. Knops, L.S. Lannes, A.S. MacDougall, R.L. McCulley, R.M. Mitchell, J.L. Moore, J.W. Morgan, B. Mortensen, H. Olde Venterink, P.L. Peri, S.A. Power, S.M. Prober, C. Roscher, M. Sankaran, E.W. Seabloom, M.D. Smith, C. Stevens, L.L. Sullivan, M. Tedder, G.F. Veen, R. Virtanen, and G.M. Wardle. 2023. Compositional variation in grassland plant communities. *Ecosphere* 14(6):e4542. <https://doi.org/10.1002/ecs2.4542>.

- Baselga, A. 2010. Partitioning the turnover and nestedness components of beta diversity. *Global Ecology and Biogeography* 19:134-143.
- Baselga, A. 2012. The relationship between species replacement, dissimilarity derived from nestedness, and nestedness. *Global Ecology and Biogeography* 21:1223-1232.
- Baselga, A. 2013. Separating the two components of abundance-based dissimilarity: balanced changes in abundance vs. abundance gradients. *Methods in Ecology and Evolution* 4:552-557.
- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Boyce, R.L., and P.C. Ellison. 2001. Choosing the best similarity index when performing fuzzy set ordination on binary data. *Journal of Vegetation Science* 12:711-720.
- Bray, J.R., and J.T. Curtis. 1957. An ordination of the upland forest communities of southern Wisconsin. *Ecological Monographs* 27:325-349.
- Chao, A., R.L. Chazdon, R.K. Colwell, and T-J. Shen. 2005. A new statistical approach for assessing similarity of species composition with incidence and abundance data. *Ecology Letters* 8:148-159.
- Chen, J., K. Bittinger, E.S. Charlson, C. Hoffmann, J. Lewis, G.D. Wu, R.G. Collman, F.D. Bushman, and H. Li. 2012. Associating microbiome composition with environmental covariates using generalized UniFrac distances. *Bioinformatics*. 28(16):2106-2113. doi:10.1093/bioinformatics/bts342
- Clarke, K.R., P.J. Somerfield, and M.G. Chapman. 2006. On resemblance measures for ecological studies, including taxonomic dissimilarities and a zero-adjusted Bray-Curtis coefficient for denuded assemblages. *Journal of Experimental Marine Biology and Ecology* 330:55-80.
- De'ath, G. 1999. Extended dissimilarity: a method of robust estimation of ecological distances from high beta diversity data. *Plant Ecology* 144:191-199.
- Faith, D.P., P.R. Minchin, and L. Belbin. 1987. Compositional dissimilarity as a robust measure of ecological distance. *Vegetatio* 69:57-68.
- Goslee, S.C., and D.L. Urban. 2007. The ecodist package for dissimilarity-based analysis of ecological data. *Journal of Statistical Software* 22(7):1-19.
- Gower, J.C. 1971. A general coefficient of similarity and some of its properties. *Biometrics* 27:857-874.
- Greenacre, M., and R. Primicerio. 2013. Measures of distance between samples: Non-Euclidean. Ch. 5 in *Multivariate analysis of ecological data*. Fundación BBVA. <http://www.multivariatestatistics.org/publications.html>
- Jaccard, P. 1912. The distribution of the flora in the alpine zone. *The New Phytologist* 11:37-50.
- Legendre, P., and M. De Cáceres. 2013. Beta diversity as the variance of community data: dissimilarity coefficients and partitioning. *Ecology Letters* 16:951-963.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. 3rd English Edition. Elsevier, Amsterdam, The Netherlands.
- Lozupone, C., and R. Knight. 2005. UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology* 71(12):8228-8235. doi:10.1128/AEM.71.12.8228-8235.2005.
- Lozupone, C., M. Hamady, S.T. Kelley, and R. Knight. 2007. Quantitative and qualitative β diversity measures lead to different insights into factors that structure microbial communities. *Applied and Environmental Microbiology* 73(5):1576-1585. doi:10.1128/AEM.01996-06
- Mahalanobis, P.C. 1936. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India* 2(1):49-55.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Plantinga, A.M., J. Chen, R.R. Jenq, and M.C. Wu. 2019. pldist: ecological dissimilarities for paired and longitudinal microbiome association analysis. *Bioinformatics* 35(19):3567-3575. doi: 10.1093/bioinformatics/btz120

Sørensen, T.J. 1948. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. *Kongelige Danske videnskabernes selskab* 5(4):1-34. [need to verify citation]

13. Multivariate Outlier Analysis

Learning Objectives

To survey visual and numerical methods of identifying multivariate outliers.
To continue using R.

Key Packages

```
require(tidyverse, vegan, labdsv)
```

Introduction

Univariate and bivariate outliers are relatively easy to detect (e.g., Zuur et al. 2010), but multivariate outliers are much more difficult. For example, if a dataset contains more than 3 variables, multivariate outliers cannot be detected graphically without first applying some method to reduce the dimensionality of the data. Creating a distance matrix is one way of doing so. If a single sample unit is a multivariate outlier, then there will be a large distance between it and all other elements.

The choice of distance measure will affect the ability to detect outliers. As an extreme example, an outlier in abundance would not be detectable if the data were summarized using a measure based on presence/absence.

Multivariate outliers can be explored visually and numerically. We'll use a hypothetical but realistic example to illustrate an outlier. We begin by opening our R Project and loading our Oak data:

```
Oak <- read.csv("data/Oak_data_47x216.csv", header = TRUE, row.names = 1)
Oak_species <- read.csv("data/Oak_species_189x5.csv", header = TRUE)
```

We'll focus on the geographic coordinates of the stands. Let's calculate the Euclidean distance among the stands:

```
require(tidyverse); require(vegan)
Oak$Stand <- row.names(Oak)
original.dis <- Oak |>
  dplyr::select(LatAppx, LongAppx) |>
  vegdist(method = "euclidean")
```


What if the decimal place is off by one position in one of the latitude values? To create this example dataset, we'll change the first latitude value and then calculate a new distance matrix. Note that this is a temporary (scripted) adjustment – since we aren't saving the changed value within oak, our real data aren't affected.

```
outlier.dis <- oak |>
  mutate(LatAppx = case_when(Stand == "Stand01" ~ LatAppx / 10,
    TRUE ~ LatAppx)) |>
  dplyr::select(LatAppx, LongAppx) |>
  vegdist(method = "euclidean")
```

Heat Maps (`coldiss()`)

Since a distance matrix contains so many values and is difficult to view, several ways have been developed to summarize it graphically. Borcard et al. (2018) provide a function, `coldiss.R`, to generate heat maps of a distance matrix. The function is available in the collection of data, functions, and scripts that they have published on their website. I have copied it to the course GitHub repository for ease of access.

To use this function:

- Create a 'functions' subfolder in your project folder.
- Download and save `coldiss.R` to this subfolder.
- Open your R project.
- Load this function using the `source()` function: `source("functions/coldiss.R")`

Note that this function is now listed in the Environment panel. There are four arguments:

- `D` – distance matrix to use.
- `nc` – number of colors to use in heat map. Default is to use 4 colors (`nc = 4`).
- `byrank` – whether to use equal-sized classes (`byrank = TRUE`) or to have equal spacing among classes (`byrank = FALSE`). For example, imagine that there are 100 distances that those distances range from 0 to 1. Setting `byrank = TRUE` means that 25 distances will be in each class; this is the default. Setting `byrank = FALSE` means that distances from 0-0.25 will be in one class, 0.25-0.5 in another class, etc.
- `diag` – whether to print object labels on the diagonal (`diag = TRUE`). Default is to not do this.

We will apply this function to our `outlier.dis` distance matrix created above. Since we are concerned here about the actual values of the distances, we will specify equal spacing among classes:

```
coldiss(outlier.dis, byrank = FALSE)
```

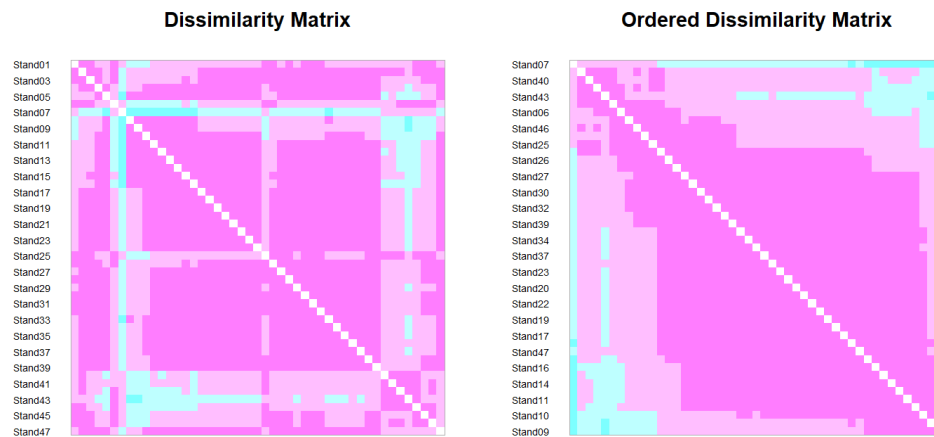


Heat map showing Euclidean distances among sample units when Stand01 is an outlier. The distances are separated into four equally spaced classes, but the first stand is such an outlier that distances between it and all other stands are in one class while distances between any other pairs of stands are in the fourth class. There are no distances in the second or third classes.

Bluer colors indicate pairs of plots that are far from one another, and pinker colors indicate pairs of plots that are closer to one another. The left-hand image shows the stands as ordered in the distance matrix, while the right-hand image shows the stands after ordering them so that similar stands are grouped together.

For comparison, here is the heat map based on the correct distances among stands:

```
coldiss(original.dis, byrank = FALSE)
```



Heat map showing Euclidean distances among sample units when all data are correct. The distances are separated into four equally spaced classes, each of which is present in the heat map.

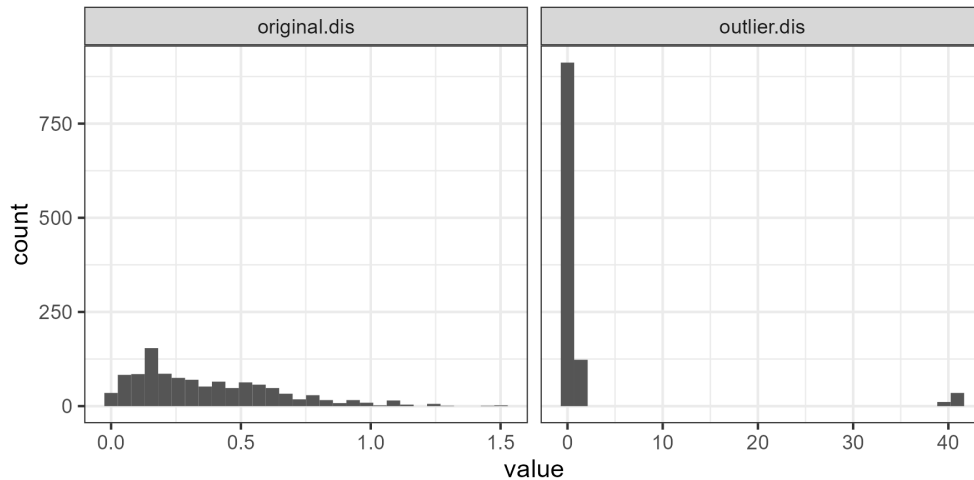
The difference between these two distance matrices can also be viewed by viewing each distance matrix as a vector of numbers. We'll combine these together so that we can graph them below:

```
distance.comparison <- data.frame(
  outlier.dis = as.numeric(outlier.dis),
  original.dis = as.numeric(original.dis) ) |>
pivot_longer(cols = c(outlier.dis, original.dis) )
```

Be sure you understand the dimensionality of this object.

Now, graph each set of distances:

```
ggplot(data = distance.comparison, aes(x = value)) +
  geom_histogram() +
  facet_grid(cols = vars(name), scales = "free_x") +
  theme_bw()
```



Pairwise distances between stands as calculated with the correct data (original.dis) and the data with an outlier (outlier.dis).

Dissimilarity Analysis (`labdsv::disana()`)

The `disana()` function from `labdsv` provides a graphical and numerical summary of a distance matrix. It produces a series of three graphs summarizing the distribution of values within the matrix:

1. Dissimilarity values, ranked from lowest to highest. Do you understand why there are so many data points on this graph? (hint: what is the formula for determining how many unique pairwise combinations there are?)
2. Minimum, mean, and maximum dissimilarity values for each sample unit.
3. Mean dissimilarity as a function of minimum dissimilarity. Note that you are asked in the Console if you want to identify individual plots. If you enter 'Y', the pointer changes to cross-hairs and you can click on data points in the graph. When done, press 'Esc' or choose 'Finish' to break out of the plot ID routine. The names of the points that you chose will be added to the graph.

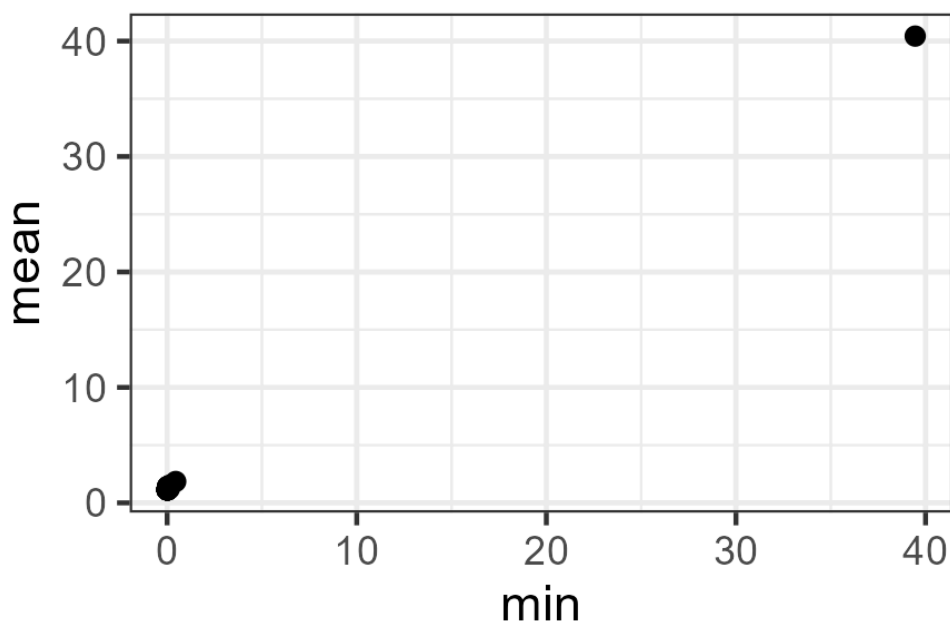
The results of `disana()` can also be saved to an object. This object is of class 'list'. It contains the minimum, mean, maximum dissimilarity of each plot to all others, as well as a vector identifying any samples selected in the last graph. You can inspect the structure of this object using the `str()` function, and can also extract individual components for further analysis or use.

Applying this to our example:

```
outliers <- disana(outlier.dis)
```

To illustrate how the components of this object can be used, let's duplicate the last plot created by `disana()` – the minimum dissimilarity compared to the minimum dissimilarity.

```
merge(outliers$min, outliers$mean, by = "row.names") |>
  rename(min = x, mean = y) |>
  ggplot(aes(x = min, y = mean)) +
    geom_point() +
    theme_bw()
```



Minimum distance (x-axis) vs. average distance (y-axis) between each stand and all other stands, where one stand is an outlier.

Note how strongly the outlier stands out from all of the others.

For comparison, apply `disana()` to the original distance matrix and then duplicate the above graphic.

Mahalanobis Distance

The Mahalanobis distance is a measure of how likely it is that an individual belongs to a group. In Section 5.3, Manly & Navarro Alberto (2017) describe how this can be done – their script is available in the book's online resources. Several of the functions outlined in the 'Common Distance Measures' chapter can calculate Mahalanobis distances, but I haven't investigated this fully. In particular, the conventional Mahalanobis distance was developed for multivariate normal data but it would also be possible to modify this technique so it is applicable to other distributional forms, such as by calculating the distance between each plot and the centroid ('center of mass') of the rest of the sample units rather than the multivariate mean.

Jackson & Chen (2004) reported that a method based on the calculation of minimum volume ellipsoids worked better than Mahalanobis distances, though I haven't seen (or looked for) a function to do this in R. Another option is to write a function in which the mean distance from one sample

unit to all others is calculated and compared to the mean distance among all other sample units. If a sample unit is an outlier, the mean distance from it to the others will be larger than the mean distance among other sample units.

Conclusions

Multivariate outliers should be investigated further to see why they are outliers. If a sample differs strongly from others in a single variable, does it reflect a data entry error that needs to be corrected? If a sample differ from others with respect to multiple variables, was it perhaps assigned to the wrong group? You as the investigator ultimately have to decide how to deal with outliers and whether or not to include them in your analyses.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Jackson, D.A., and Y. Chen. 2004. Robust principal component analysis and outlier detection with ecological data. *Environmetrics* 15:129-139.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- Zuur, A.F., E.N. Ieno, and C.S. Elphick. 2010. A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution* 1:3-14.

Media Attributions

- coldiss_outlier.dis
- coldiss_original.dis
- distance.comparison
- disana

PART II

GROUP COMPARISONS

This section explores concepts related to multivariate statistical tests of differences between groups. Concepts are presented in short chapters so that the reader can easily jump to the information they seek.

We begin by considering ANOVA / MANOVA, which illustrate how variation can be partitioned and how that partitioning can be expressed in a test statistic.

ANOVA and MANOVA assess the significance of the test statistic via its expected distribution – this is why they require parametric assumptions such as normality. The techniques we will cover are **permutation-based**, which means that they do not make these assumptions. We'll consider:

- General principles about permutation tests,
- How to control permutations,
- How to restrict permutations.

The techniques we'll cover as also **distance-based**, meaning that they are based on the distance matrix calculated from the response(s). The techniques that we'll cover are:

- ANalysis Of SIMilarity (ANOSIM)
- Mantel tests
- Multi-Response Permutation Procedure (MRPP)
- PERMutational ANOVA/MANOVA (PERMANOVA)
- PERMutational analysis of DISPersion (PERMDISP)
- Randomization of Residuals in a Permutation Procedure (RRPP)
- **MVABUND**

We'll illustrate each test using two datasets, a simple one that can be worked by hand and a larger one from our oak plant communities that requires computer calculations. Included is a script to load and process the larger dataset.

Most of the techniques are designed to test for differences among *a priori* groups such as experimental treatments. Some of the techniques are also appropriate for testing linear relationships with continuously distributed explanatory variables, as in regression. We'll also consider how to analyze complex models.

Finally, I provide a comparison of the techniques.

14. ANOVA / MANOVA

Learning Objectives

To review the structure of ANOVA and MANOVA.

To review the assumptions of ANOVA and MANOVA when testing for differences among *a priori* groups.

Introduction

Analysis of variance (ANOVA) is an extremely popular and versatile technique for univariate analyses. Multivariate analysis of variance (MANOVA) is less commonly used and more restrictive in its assumptions. However, the structure of these techniques is a helpful starting point for the techniques we're focusing on.

Assumptions of ANOVA and MANOVA

ANOVA and MANOVA are parametric techniques, meaning that they make assumptions about the parameters (distributional form) of the population from which samples are taken. For our purposes here, the key assumptions of ANOVA are that:

1. Errors are normally distributed
2. Variances are equal among groups

With MANOVA, these assumptions are more restrictive:

1. Errors conform to multivariate normality (i.e., are normally distributed in all dimensions)
2. The entire variance-covariance matrix is homogeneous (i.e., variances of all variables and covariances between all pairs of variables are equal across all groups)

(See the discussion about exchangeable units for another assumption, independence, that is relevant for both parametric and permutation tests.)

ANOVA is optimal and preferred for data that meet these assumptions but can be far from optimal when data do not meet them. Of course, it is also limited to univariate responses.

MANOVA is designed for multivariate responses, but is even more sensitive than ANOVA to deviations from its assumptions. It therefore is often inappropriate for community-level ecological data.

Key Takeaways

ANOVA is a method of partitioning variation to different sources and expressing those patterns in a test statistic. Many other techniques have a similar structure.

Structure of ANOVA (and MANOVA)

Although ANOVA is not optimal for all types of data, its basic structure remains insightful in two ways:

1. As an example of how to partition variation
2. As an example of how a test statistic can represent that partitioning

Partitioning Variation

Recall that ANOVA focuses on the variation within a dataset. In a simple one-way ANOVA, three sources of variation are distinguished:

- SST: total variation within data (i.e., total sum of squares), calculated as the sum of the squared deviations between every observation and the grand mean of the data
- SSB: variation between groups
- SSW: variation within groups

These sources of variability are related:
 $SST = SSB + SSW$

Verbally, we have determined how much of the total variation (SST) is due to differences between the groups (SSB) and how much of it is due to variation within groups (SSW). The variation within the groups is the residual or unexplained variation.

MANOVA partitions the variation similarly to ANOVA, except that it accounts for both the variance within variables and the covariance between pairs of variables.

This idea of partitioning variability is central to several of the techniques we will be discussing.

Every Test Requires a Test Statistic

The ANOVA test statistic describes how the variation is partitioned. The F -statistic is calculated as the ratio of variation among groups to variation within groups, weighting each term by its degrees of freedom (df):

$$F = \frac{SSB/(t - 1)}{SSW/(N - t)}$$

where t is the number of groups and N is the total sample size (note that this equation is for a balanced one-way ANOVA; the details would differ slightly for other designs).

Like ANOVA, every test requires a test statistic. Which test statistic to use is decided by those who create a test.

As a parametric test, ANOVA assesses the likelihood of this test statistic by comparing the value obtained from the actual data with the theoretical distribution of its possible values. Our tests are permutation-based and therefore will compare the test statistic with a set of possible values obtained by permuting the data.

15. Sample Datasets

Learning Objectives

To introduce two datasets for use throughout this section of the course:

- A simple dataset for making calculations by hand
- A larger dataset to illustrate how statistical tests can be applied in R

The larger dataset includes a script to automate its loading and initial data adjustments.

Key Packages

```
require(vegan, labdsv, tidyverse)
```

Throughout this section, we will work with the following two datasets. Each chapter assumes that you've loaded the data as described below.

Simple Example

This dataset is small enough that we can do calculations by hand. Seeing the calculations by hand helps clarify what happens when we apply the same techniques to larger datasets. We can also verify the calculations by repeating the analysis in R.

The data include two response variables (Resp1 and Resp2) on 6 plots, and a column identifying the group to which each plot belongs.

Sample Unit	Resp1	Resp2	Group
Plot1	1	4	A
Plot2	3	2	A
Plot3	5	3	A
Plot4	9	12	B
Plot5	10	8	B
Plot6	11	11	B

We can use a dataset like this to ask the question ‘do groups A and B differ in overall response’? Note that this is a multivariate question; we are not asking about Resp1 or Resp2 individually. Follow-up analyses could consider the individual response variables if the multivariate response is significant.

This dataset is available as the file ‘Permutation.example.csv’ from the book's GitHub repository. Download it to the 'data' folder of your course folder.

Open your R project and then load these data:

```
perm.eg <- read.csv("data/Permutation.example.csv", header = TRUE, row.names = 1)
```

It can also be manually entered:

```
perm.eg <- data.frame(
  row.names = c("Plot1", "Plot2", "Plot3", "Plot4", "Plot5", "Plot6"),
  Resp1 = c(1, 3, 5, 9, 10, 11),
  Resp2 = c(4, 2, 3, 12, 8, 11),
  Group = c("A", "A", "A", "B", "B", "B")
)
```

Here's the distance matrix for our simple example:

```
Resp.dist <- perm.eg |>
  dplyr::select(Resp1, Resp2) |>
  dist()
round(Resp.dist, 3)
```

```

      Plot1 Plot2 Plot3 Plot4 Plot5
Plot2 2.828
Plot3 4.123 2.236
Plot4 11.314 11.662 9.849
Plot5 9.849 9.220 7.071 4.123
Plot6 12.207 12.042 10.000 2.236 3.162
```

I rounded the distance matrix to 3 decimal places for display purposes; in practice I would keep all decimals as calculated by R.

Grazing Example (with a script!)

We'll also look at the oak plant community dataset. Specifically, we'll ask whether community composition is correlated with differences in current grazing status.

We begin by importing the data. Recall from the metadata that this dataset contains data from 47 stands. We'll create separate objects for the composition and explanatory data. We'll then make two adjustments to the composition data:

- Remove rare species (those present on <5% of sample units)
- Relativize by species maxima

Finally, we'll use the Bray-Curtis distance measure to calculate the distance between every pair of stands.

We've done all of these steps before. We could continue to write these steps out each time, but I've prepared a script to conduct them. The script (`load.oak.data.R`) is available in the book's GitHub repository.

Once you've saved the script to the 'scripts' sub-folder within your analysis folder and opened your R project file, call the script using `source()`:

```
source("scripts/load.oak.data.R")
```

Open the script and review it to ensure that you understand what happens in it:

- Three packages (`vegan`, `labdsv`, `tidyverse`) are loaded
- Data files are imported
- Response and explanatory variables are saved to separate objects
- Rare species are removed from compositional data
- Compositional data are relativized by species maxima
- Bray-Curtis distance matrix is calculated from the relativized compositional data
- Resulting matrix is saved as the object `oak1.dist`

If our research questions warranted other changes (e.g., relativizing by site totals), we could adjust the script to include them. The original objects are also available in RStudio if you want to just use the script to load them and then use them in other ways.

Our grouping factor for this example is current grazing status (Yes, No). We could index this factor each time we need it (`Oak$GrazCurr`) but for clarity we'll create an object consisting of just the grazing status of each plot:

```
grazing <- Oak$GrazCurr
```

This is not part of the script because the focal explanatory variables will often vary from study to study.

16. Permutation Tests

Learning Objectives

- To explore the theory behind permutation-based tests.
- To illustrate how permutation tests can be conducted in R.

Key Packages

```
require(tidyverse)
```

Introduction

A statistical test involves the calculation of a test statistic followed by an assessment of how likely the calculated value of the test statistic would be if the data were randomly distributed. In the case of ANOVA, the test statistic is the F -statistic, and it is compared to the theoretical distribution of F -values with the same degrees of freedom.

We will consider a number of tests using a range of test statistics. However, there is no theoretical distribution for these test statistics. Rather, **the distribution of the test statistic will be derived from the data**. Generally, this reference distribution is generated by permuting (i.e., randomly reordering) the group identities, recalculating the test statistic, saving that value, and repeating this process many times (Legendre & Legendre 2012).

If the patterns observed in the data are unlikely to have arisen by chance, then the actual value of the test statistic should differ from the set of values obtained from the permutations.

Key Takeaways

- The number of sample units directly affects the number of permutations.
- Some permutations are functionally equivalent, so there are fewer combinations for a given sample size.

Number of Permutations (and Combinations)

The number of samples directly affects the number of possible permutations and combinations.

A **permutation** is a re-ordering of the sample units. From a total of n sample units, the number of possible permutations (P) is:

$$P = n!$$

The number of permutations rises rapidly with sample size – see the table below. To explore this, enumerate the permutations of the letters {a, b, C, D}. There are four values, so the number of permutations is:

$$P = n! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

Assume that the letters {a, b, C, D} represent sample units, with lower and upper cases representing different groups. In other words, the first two sample units are in one group and the last two are in the other group. One permutation of these letters is {a, C, b, D}. In this permutation, we assign the first and third sample units to one group and the second and fourth sample units to the other group. Note that this 'assignment' is temporary and only for the purpose of this permutation.

When we think about group comparisons, it is helpful to recognize that some permutations are functionally equivalent. For example, consider the permutations {a, C, b, D} and {C, a, D, b}. In both permutations, the first and third sample units are assigned to one group and the second and fourth sample units are assigned to the other group. These permutations represent the same combination. A **combination** is a unique set of sample units, irrespective of sample order within each set. The number of combinations (C) of size r is:

$$C_r^n = \frac{n!}{r!(n-r)!}$$

(this equation is from Burt & Barber 1996). This is for equally-sized groups; the calculations are more complicated for groups of different sizes.

For our simple example of four sample units, there are

$$C_r^n = \frac{n!}{r!(n-r)!} = \frac{4!}{2!(4-2)!} = \frac{4 \cdot 3 \cdot 2 \cdot 1}{2 \cdot 1(2 \cdot 1)} = 6$$

The following table shows how the number of permutations and combinations (into two equally-sized groups) rises rapidly with the number of sample units.

Samples (<i>n</i>)	Permutations (<i>P</i>)	Samples per group (<i>r</i>)	Combinations (<i>C</i>)
4	24	2	6
8	40,320	4	70
12	479,001,600	6	924
16	2.1×10^{13}	8	12,870
20	2.4×10^{18}	10	184,756
24	6.2×10^{23}	12	2,704,156
28	3.0×10^{29}	14	40,116,600
32	2.6×10^{35}	16	601,080,390
36	3.7×10^{41}	18	9,075,135,300
40	8.2×10^{47}	20	137,846,528,820

Probabilities

Permutation-based probabilities are calculated as the proportion of permutations in which the computed value of the test statistic is equal to or more extreme than the actual value. This calculation can be made with any number of permutations, though it is easier to do so mentally if the denominator is a multiple of ten, such as 1,000.

The actual sequence of group identities is one of the possible permutations and therefore is included in the denominator of the probability calculation. This is why it is common to do, for example, 999 permutations – once the actual sequence is included, the denominator of the probability calculation is 1,000.

The minimum possible *P*-value is partly a function of the number of permutations. For example, consider a scenario in which the test statistic is larger when calculated with the real data than when calculated for any of the permutations. If we had only done 9 permutations, we would calculate

$$P = \frac{1}{9 + 1} = 0.1$$

However, if we had done 999 permutations, we would calculate

$$P = \frac{1}{999 + 1} = 0.001$$

Would it make sense to declare that the effect is significant in the second case but not in the first?

For studies with reasonably large sample sizes, there are many more permutations than we can reasonably consider. For example, there are 8.2×10^{47} permutations of 40 samples as reported in the above table. If we considered one every millisecond (i.e., 1000 per second), it would still take us 2.6×10^{37} years to consider them all!

Given the large number of possible permutations, we usually assess only a small fraction of them. This means that permutation-based probability estimates are subject to sampling error and vary from run to run. The variation between runs declines as the number of permutations increases; more permutations will result in more consistent estimates of the probability associated with a

test statistic. Legendre & Legendre (2012) offer the following recommendations about how many permutations to compute:

- Use 500 to 1000 permutations during exploratory data analyses
- Rerun with an increased number of permutations if the computed probability is close to the preselected significance level (either above or below)
- Use more permutations (~10,000) for final, published results

The `system.time()` function can be used to measure how long a series of permutations requires.

Key Takeaways

The statistical significance of a permutation-based test is the proportion of permutations in which the computed value of the test statistic is equal to or more extreme than the actual value.

The actual value of the test statistic is unaffected by the number of permutations.

It's ok to use small numbers of permutations during exploratory analyses, but use a large number (~10,000) for final analyses.

Exchangeable Units

I mentioned two assumptions of ANOVA / MANOVA in that chapter; a third foundational assumption of both techniques is that the sample units are **independent**. Analyses may be suspect when this is not the case or when the lack of independence is not properly accounted for. Failure to account for lack of independence is a type of pseudoreplication (Hurlbert 1984).

The assumption of independence also applies for permutation tests – it is what justifies exchangeability in a permutation test. This means that it is possible to analyze a permutation-based test incorrectly. Permutations need to be restricted when sample units are not exchangeable. The correct way of permuting data depends on the structure of the study and the hypotheses being tested. The basic idea is that **the exchangeable units that would form the denominator when testing a term in a conventional ANOVA are those that should be permuted during a permutation test of that term.**

Questions about independence and exchangeability are particularly pertinent for data obtained from complex designs that include multiple explanatory variables simultaneously. See Anderson & ter Braak (2003), Anderson et al. (2008), and Legendre & Legendre (2012) for details on how to identify the correct exchangeable units for a permutation test.

For example, in a split-plot design one factor can be applied to whole plots and another factor to split plots (i.e., within the whole plots). Each of these factors would require a different error term.

- Analyses of the whole plot factor use the unexplained variation among whole plots as the error term. This is evident in the fact that the df for the whole plot error term is based on the number of whole plots, regardless of how many measurements were made within them. In a permutation test, variation among whole plots is assessed by restricting permutations such that all observations from the same whole plot are permuted together. Variation within whole

plots is ignored when analyzing whole plot effects.

- Analyses of the split-plot factor use the residual as the error term and therefore do not require restricted permutations. However, they do require the inclusion of a term that uniquely identifies each whole plot so that the variation among whole plots is accounted for. Doing so allows the analysis to focus on the variation within whole plots. If a model included interactions with the split-plot factor, these would also be tested at this scale.

More information on this topic is provided in the chapter about complex models.

Implementation in R

The `sample()` function can be used to permute data. If needed, you can use the `size` argument to create a subset, and the `replace` argument to specify whether to sample with replacement (by default, this is `FALSE`).

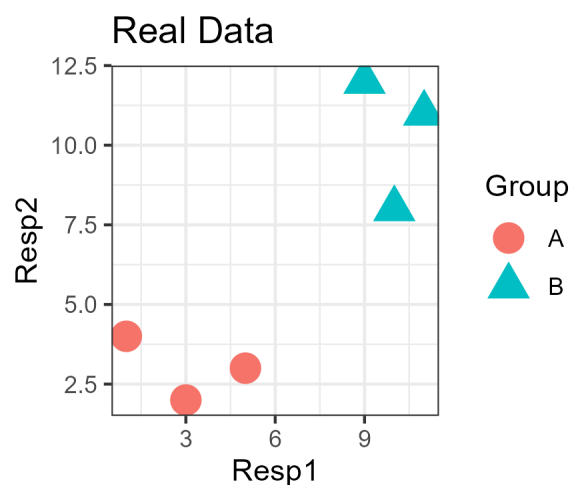
The `vegan` package, drawing on the `permut` package, includes a number of options for conducting permutations. This topic is explained in more detail in the chapters about controlling permutations and restricting permutations.

Simple Example, Graphically

Since our simple example only has two response variables, it is easily visualized:

```
library(tidyverse)
ggplot(data = perm.eg, aes(x = Resp1, y = Resp2)) +
  geom_point(aes(colour = Group, shape = Group), size = 5) +
  labs(title = "Real Data") +
  theme_bw()
ggsave("graphics/main.png", width = 3, height = 2.5, units = "in", dpi = 300)
```

(did you notice that we saved the image, and where we saved it?)



To conduct a permutation, we can permute either the grouping factor or the data. Can you see why these are equivalent? We would not permute both at the same time ... do you see why that is?

We'll permute the grouping factor:

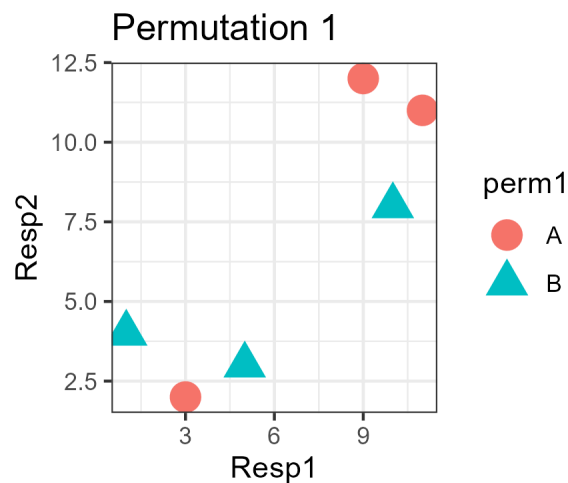
```
perm.eg$perm1 <- sample(perm.eg$Group)
```

We've added the permutation as a new column within the `perm.eg` object. View the object to compare the permutation with the original grouping factor. Note that the number of occurrences of each group remains the same in permutations as in the original.

Let's visualize this permutation of the data. We can use the same code as above with a few changes:

- Name of column identifying the groups used for colour and shape within `geom_point()`.
- Title of figure
- Name of file to which image is saved

```
ggplot(data = perm.eg, aes(x = Resp1, y = Resp2)) +  
  geom_point(aes(colour = perm1, shape = perm1), size = 5) +  
  labs(title = "Permutation 1") +  
  theme_bw()  
ggsave("graphics/perm1.png", width = 3, height = 2.5, units = "in", dpi = 300)
```



It is possible but somewhat unlikely that the group identities in your graph match this one. Be sure you understand why!

Simple Example, Distance Matrix

Analyses will be based on the distance matrix so let's consider this. Here it is:

```
Resp.dist <- perm.eg |>  
  dplyr::select(Resp1, Resp2) |>  
  dist()  
round(Resp.dist, 3)
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	2.828				
Plot3	4.123	2.236			
Plot4	11.314	11.662	9.849		
Plot5	9.849	9.220	7.071	4.123	
Plot6	12.207	12.042	10.000	2.236	3.162

Group identity was not part of the distance matrix calculation. This means that permuting the group identities doesn't change the distance matrix itself.

Permuting group identities does change which distances connect sample units assigned to the same group. For example, plots 1, 2, and 3 are all in group A in our real data but plots 2, 4, and 6 were assigned to group A in Permutation 1 above.

Note: To keep our example simple, we did not relativize the data and calculated Euclidean distances. These decisions do not affect the permutations but, as discussed before, these decisions should be based on the nature of your data and your research questions.

Grazing Example

Our grouping factor for this example is current grazing status (Yes, No). The `sample()` function can also be applied here:

```
sample(grazing)
```

We'll use this example in more detail in upcoming chapters.

References

Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK. 214 p.

Anderson, M.J., and C.J.F. ter Braak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.

Burt, J.E., and G.M. Barber. 1996. *Elementary Statistics for Geographers*. 2nd edition. Guilford Publications.

Hurlbert, S.H. 1984. Pseudoreplication and the design of ecological field experiments. *Ecological Monographs* 54:187-211.

Legendre, P., and L. Legendre. 2012. *Numerical ecology*. 3rd English edition. Elsevier, Amsterdam, The Netherlands.

Media Attributions

- main

- perm1

17. ANOSIM

Learning Objectives

- To understand the theory behind ANOSIM.
- To apply ANOSIM manually and through R.

Key Packages

```
require(vegan)
```

Theory

ANalysis Of SIMilarity (ANOSIM) was proposed by Clarke & Green (1988). It can be used to determine whether there are statistically significant differences between two or more groups. ANOSIM is a non-parametric technique based on ranks. You may have encountered non-parametric univariate techniques based on ranks before, such as the Kruskal-Wallis test or the Mann-Whitney U test. We will encounter ranked data again in a few weeks when we discuss non-metric multidimensional scaling (NMDS).

The key idea behind ANOSIM is that if the grouping variable is important, then on average the rank distance within groups will be smaller than the rank distance between sample units from different groups.

Recent ecological examples are provided by Muthukrishnan et al. (2019), who compared marine microbial communities, and Sun et al. (2019), who compared soil fungal assemblages beneath different shrubs.

ANOSIM has lower power (higher probability of Type II statistical error) if strong gradients are present in the data (Gotelli & Ellison 2004). See the end of this chapter for recent developments with this technique.

As available in R, ANOSIM is limited in its utility as it can only be used for one-way and fully crossed or nested two-way designs.

Key Takeaways

ANOSIM is a non-parametric test based on the rank distances among sample units. If a grouping variable is important, the mean rank distance among sample units within a group will be smaller than the rank distance between sample units from different groups.

The test statistic (R) ranges from -1 (all lowest ranks are between groups – an unusual situation) to +1 (all lowest ranks are within groups).

A generalized ANOSIM statistic can be applied to complex designs with 2 or 3 factors.

Basic Procedure

The basic procedure for ANOSIM is as follows:

1. Begin with a distance matrix. Convert the distances within the matrix to their ranks, such that the smallest distance has a rank of 1.
2. Group the ranks to distinguish those that represent observations within the same group and those that represent observations from different groups.
3. Calculate the average rank within groups (\bar{r}_W) and the average rank between groups (\bar{r}_B). If the grouping factor is important, the mean rank within groups should be smaller than the mean rank between groups. Can you see why this is so?
4. Calculate and save the ANOSIM test statistic, R (confusing; not the software!):

$$R = \frac{\bar{r}_B - \bar{r}_W}{n(n-1)/4}$$

R ranges from -1 if all of the lowest ranks are between groups to +1 if all of the lowest ranks are within groups. It is zero if the high and low ranks are perfectly mixed between and within groups.

5. Test the significance of R via permutations:
 - Shuffle the order of the values in the grouping factor.
 - Recalculate R using the permuted grouping factor (i.e., assuming that each observation came from the group specified in the permuted grouping factor). Save this value.
 - Repeat: reshuffle and recalculate the specified number of times. The permutations produce a sampling distribution of R against which the actual test statistic is compared.
 - Calculate the P -value as the proportion of permutations that yielded an R equal to or stronger than the R calculated from the actual data.

Simple Example, Worked by Hand

We'll start with the dissimilarity matrix based on the two response variables:

```
Resp.dist <- dist(perm.eg[ , c("Resp1", "Resp2")])
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	2.828				
Plot3	4.123	2.236			
Plot4	11.314	11.662	9.849		
Plot5	9.849	9.220	7.071	4.123	
Plot6	12.207	12.042	10.000	2.236	3.162

For clarity, distances between plots from different groups are shown in bold (compare with the group identities to verify this).

Now, we replace the distances with their ranks. Tied values are given the average of the two ranks. The distances between plots from different groups are again shown in bold.

	Plot1	Plot	Plot3	Plot4	Plot5
Plot2	3				
Plot3	5.5	1.5			
Plot4	12	13	9.5		
Plot5	9.5	8	7	5.5	
Plot6	15	14	11	1.5	4

We can use this to calculate the average rank within and between groups:

- Within groups: $\bar{r}_W = \frac{3 + 5.5 + 1.5 + 5.5 + 1.5 + 4}{6} = 3.5$
- Between groups: $\bar{r}_B = \frac{12 + 9.5 + 15 + 13 + 8 + 14 + 9.5 + 7 + 11}{9} = 11$

Calculate R :

$$R = \frac{\bar{r}_B - \bar{r}_W}{n(n-1)/4} = \frac{11 - 3.5}{6(6-1)/4} = \frac{7.5}{7.5} = 1$$

Inspection of the ranked distance matrix shows that every small rank is among observations in the same group, which is why R is at its maximum possible value. The statistical significance of this statistic would be assessed with a permutation test.

Implementation in R (**vegan::anosim()**)

In R, ANOSIM can be conducted using the `anosim()` function in the `vegan` package. The usage of this function is:

```
anosim(
  x,
  grouping,
```

```

permutations = 999,
distance = "bray",
strata = NULL,
parallel = getOption("mc.cores")
)

```

The arguments are:

- **x** – an object of class matrix or data frame in which rows are samples and columns are response variable(s), or a dissimilarity object (class `dist`) or a symmetric square matrix of dissimilarities
- **grouping** – factor for grouping observations
- **permutations** – number of permutations to conduct to assess the significance of *R*, or a list of permutation instructions obtained using the `how()` function. This latter option is necessary with complex designs where permutations need to be restricted – see this chapter for details. Default is to conduct 999 permutations without any restrictions to the permutations.
- **distance** – distance measure applied to object if it is not a distance matrix. The `vegdist()` function is used here, along with its default distance measure, Bray-Curtis. However, any distance measure available through `vegdist()` can be specified.
- **strata** – integer vector or factor identifying strata that permutations are to be restricted within.
- **parallel** – option for parallel processing

If saved, the resulting object is of class 'anosim'. Objects of this class have pre-defined methods for applying the `plot()` and `summary()` functions.

Simple Example, in R

Applying ANOSIM to the distance matrix:

```

simple.results.anosim <- anosim(x = Resp.dist, grouping = perm.eg$Group,
permutations = 999)
simple.results.anosim

```

```

Call:
anosim(x = Resp.dist, grouping = perm.eg$Group, permutations = 999)
Dissimilarity: euclidean

ANOSIM statistic R: 1
  Significance: 0.1Permutation: free
Number of permutations: 719

```

Do the two groups differ in overall response?

Note the messages that were displayed on the screen while conducting the analysis:

```

'nperm' >= set of all permutations: complete enumeration.
Set of permutations < 'minperm'. Generating entire set.

```

This is because we specified 999 permutations but there are only 720 possible permutations of 6 sample units ($n!$). The `permutations` argument calls another function which calculates how many permutations are possible. If more were specified than are possible, the entire set of permutations is calculated and evaluated.

Side notes:

1. Other packages may allow more permutations. In these cases, some of the permutations would be duplicates. This is ok – since the P -value is a proportion it is minimally affected by how much and which permutations are duplicated – though computationally inefficient.
2. It may seem logical to specify that we want 720 permutations since that's how many are possible, but we do not control the identity of the permutations themselves. For example, a set of permutations could include some duplicates but not include all possible permutations. Therefore, it is preferable to let the function compare the requested and possible permutations.

Grazing Example, in R

Applying ANOSIM to our grazing example:

```
grazing.results.anosim <- anosim(x = Oak1.dist, grouping = grazing, permutations = 999)
grazing.results.anosim
```

```
Call:
anosim(x = Oak1.dist, grouping = grazing, permutations = 999)
Dissimilarity: bray

ANOSIM statistic R: 0.1333
      Significance: 0.015

Permutation: free
Number of permutations: 999
```

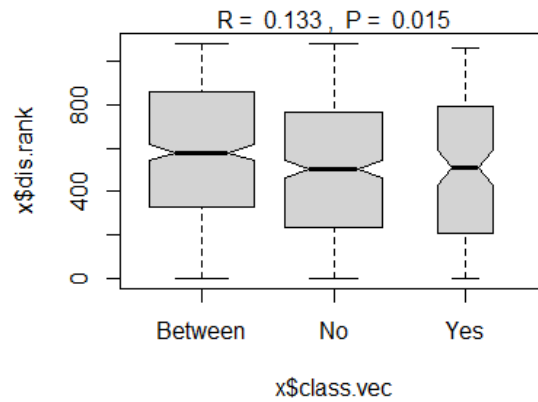
Is there a relationship between current grazing status and community composition?

The P -value is the proportion of permutations that resulted in a value of R as large or larger than that calculated using the actual grouping factor. Note that the P -value differs slightly among individual runs of this test due to differences in which permutations were used.

Rerun this test with differing numbers of permutations (9, 99, 999, 9999). How does the P -value change? Does the correlation itself change?

The `plot()` function generates different products based on the class of the object to which it is applied. When applied to an object of class 'anosim':

```
plot(grazing.results.anosim)
```



Box-and-whisker plots showing the ranks of distances among pairs of sample units where those sample units are in different groups ('Between'), both in the 'No' group, or both in the 'Yes' group.

More detailed results from this analysis can be viewed with the `summary()` function:
`summary(grazing.results.anosim)`

```
Call:
anosim(x = Oak1.dist, grouping = grazing, permutations = 999)
Dissimilarity: bray

ANOSIM statistic R: 0.1333
Significance: 0.015

Permutation: free
Number of permutations: 999

Upper quantiles of permutations (null model):
 90%   95%  97.5%   99%
0.0653 0.0906 0.1159 0.1390

Dissimilarity ranks between and within classes:
      0%   25%  50%   75% 100%   N
Between 2 328.75 580 859.75 1079 510
No      1 238.00 504 766.50 1081 435
Yes     3 211.50 509 793.25 1058 136
```

The `summary()` function does not display all of the results from this analysis. More details can be seen by investigating the structure of the results object:
`str(grazing.results.anosim)`

Items within the results object can be individually indexed and used in other functions. For example:

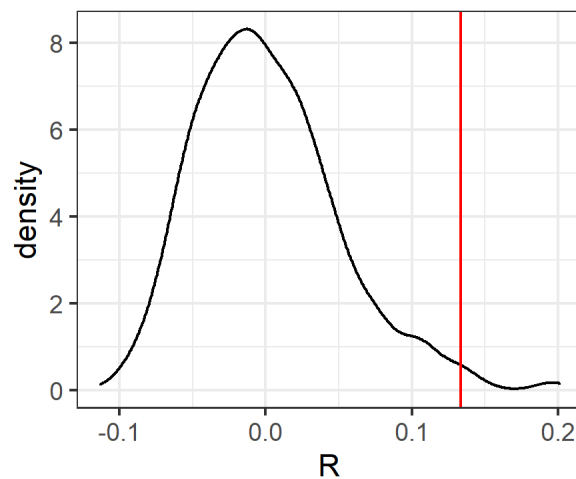
```
grazing.results.anosim$statistic # The test statistic R
grazing.results.anosim$perm      # Values from permutations
```

Let's look at how the test statistic compares with the distribution of values obtained from permutations. To visualize this, we first need to combine the calculated values of the statistic and save them in a data frame.

```
R.values <- with(grazing.results.anosim, data.frame(R = c(statistic, perm) ) )
R.values$Type <- c("actual", rep("perm", length(R.values$R) - 1))
```

Now, we can plot these data. We'll create a density estimate (smoothed histogram) of the values of the test statistic as obtained from the permutations, and superimpose onto it a vertical line showing the value obtained with the real data.

```
ggplot(data = R.values, aes(x = R)) +
  geom_density() + # permuted P-values
  geom_vline(data = R.values[R.values$Type == "actual" , ],
    aes(xintercept = R), colour = "red") +
  theme_bw()
ggsave("graphics/ANOSIM.R.png", width = 3, height = 2.5, units = "in", dpi = 300)
```



Density distribution of values of R test-statistic obtained from 1000 permutations of the Oak compositional data when testing for an effect of current grazing status on species composition. The red line is the observed value of R. The P-value is the area to the right of the red line (number of permutations that yielded equal or larger values of R) divided by the total area under the curve (total number of permutations).

Recent ANOSIM Developments

A Generalized ANOSIM Statistic

Recent work has proposed a generalized ANOSIM statistic, R° . It is defined as “the slope of the linear regression of ranked resemblances from observations against ranked distances among samples, the latter from a simple model matrix assigning the values 1 and 0 to between- and within-group distances, respectively” (Somerfield et al. (2021a, p. 901). If applied to a simple design (e.g., a single factor with two levels, as in our grazing example), it returns the same test statistic as the original R statistic. However, it can also be applied to more complex designs (see below). This test statistic is still permutation-based and, when applied to complex designs, the permutation procedures need to reflect those designs.

The generalized R° statistic is still a correlation with a maximum value of 1. It achieves its maximum value if “all dissimilarities between groups are larger than any within groups ... and all dissimilarities between groups which are placed further apart in the model matrix are larger than any dissimilarities between groups which the model puts closer together” (Somerfield et al. 2021a, p.906).

Grazing Example

We’ll illustrate the calculation of this generalized R° statistic with our grazing example.

To begin, we have to create a distance matrix tracking whether pairs of stands are in the same grazing treatment or not.

```
graz.dist <- as.numeric( as.factor( grazing ) ) |>
dist()
```

Verify that you understand why this object contains the values it contains.

Next, we combine the Bray-Curtis distances between sample units (i.e., compositional dissimilarities) and the distances based on grazing treatment.

```
Oak.eg <- data.frame(
  Resp = Oak1.dist,
  Group = graz.dist )
```

Make sure you understand how many rows this object has, and why.

Next, we convert each column of distances into ranks:

```
Oak.eg <- Oak.eg |>
  mutate(Resp.rank = rank(Resp),
         Group.rank = rank(Group))
```

```
head(Oak.eg)
```

Resp	Group	Resp.rank	Group.rank
------	-------	-----------	------------

1	0.3130190	1	2	826.5
2	0.4395344	0	9	286.0
3	0.5988716	1	193	826.5
4	0.8355293	1	965	826.5
5	0.5622599	0	126	286.0
6	0.4604209	0	15	286.0

Each of the new columns is a set of ranks. There are many repeated values due to ties – for example all ranks in **Group** were either 0 or 1 so there are only two values in **Group.rank**.

Now, fit a linear model to the ranks:

```
lm(Resp.rank ~ Group.rank, data = Oak.eg)
```

```
Call:
lm(formula = Resp.rank ~ Group.rank, data = Oak.eg)

Coefficients:
(Intercept)  Group.rank
  468.8704      0.1333
```

The slope of this model is the same value as the ANOSIM test statistic that we calculated earlier (`grazing.results.anosim$statistic`).

It may also be helpful to graph these ranks against one another, and fit this line to the data.

Extensions of ANOSIM

Somerfield et al. (2021a, b, c) published a series of related articles explaining how this generalized ANOSIM can be applied in a variety of settings.

Ordered factors – an ordinal factor with more than two levels. For example, samples along an environmental gradient (distance from oil rig: > 3.5 km, 1-3.5 km, 0.25-1 km, or < 0.25 km), or times along a time series (sites representing a long-term gradient in heavy metal contamination) (Somerfield et al. (2021a).

Two-factor designs – including nested and crossed designs, with or without ordered factors, and with or without subsampling (Somerfield et al. 2021b). This article notes that ANOSIM is fully non-parametric and cannot test interactions; instead it “provides a robust, comparable and globally interpretable measure of magnitude of overall community change associated with each factor, having excised any possible effect from the factor(s) it is crossed with, irrespective of whether the factors interact or not” (p. 911). Their Table 1 provides details for four 1-factor designs and fourteen 2-factor designs, with examples. It is reprinted in Appendix 2.

Three-factor designs – this is an extension of the 2-way design to include a third factor (Somerfield et al. 2021c). Their Table 1 provides details for thirteen 3-factor designs, with examples. It is reprinted in Appendix 2.

These extensions highlight the utility of the generalized ANOSIM for ordered factors and note that analysis of a genuinely ordered factor will be more powerful if that ordering is incorporated into

the test design (but conversely analyses would be less powerful if applied to genuinely unordered factors).

Unfortunately, to my knowledge these extensions are not currently available in R. The examples provided in the articles cited here were conducted in PRIMER, a commercially available software package.

The design considerations highlighted in these tables are also relevant for other techniques.

References

Clarke K.R., and R.H. Green. 1988. Statistical design and analysis for a 'biological effects' study. *Marine Ecology Progress Series* 46:213-226.

Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.

Muthukrishnan, T., M. Al Khaburi, and R.M.M. Abed. 2019. Fouling microbial communities on plastics compared with wood and steel: are they substrate- or location-specific? *Microbial Ecology* 78:361-379.

Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021a. A generalized analysis of similarities (ANOSIM) statistic for designs with ordered factors. *Austral Ecology* 46:901-910.

Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021b. Analysis of similarities (ANOSIM) for 2-way layouts using a generalized ANOSIM statistic, with comparative notes on Permutational Multivariate Analysis of Variance (PERMANOVA). *Austral Ecology* 46:911-926.

Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021c. Analysis of similarities (ANOSIM) for 3-way designs. *Austral Ecology* 46:927-941.

Sun, Y., Y. Zhang, W. Feng, S. Qin, and Z. Liu. 2019. Revegetated shrub species recruit different soil fungal assemblages in a desert ecosystem. *Plant and Soil* 435:81-93.

Media Attributions

- `grazing.results.anosim`
- `ANOSIM.R`

18. Mantel Test

Learning Objectives

- To understand the theory behind Mantel tests.
- To apply Mantel tests manually and through R.

Resources

van Mantgem & Schwiik (2009)

Key Packages

```
require(vegan)
```

Theory

Mantel (1967) first described this test and used it to examine spatial patterns in the occurrence of leukemia.

The Mantel test examines the correlation between two distance matrices from the same samples. The distance matrices can be based on actual data (species abundances, environmental measurements, spatial coordinates) or hypothesized data (e.g., dummy variables coding for treatments). For example, Mantel tests are commonly used to test for spatial autocorrelation as described below.

Mantel tests assume that the matrices are independent and that the correlations are linear. Of course, since the matrices are being compared to one another, it is also essential that samples are listed in the same order in both matrices.

Mantel tests are generally used in two ways in community ecology:

- To compare two independent empirical matrices. For example, spatial autocorrelation can be

explored by comparing a distance matrix based on some response with a distance matrix based on the Euclidean distance between the sample units as determined from their spatial coordinates.

- To assess the goodness of fit between data and an *a priori* model (i.e., one not generated by this dataset). This is how a Mantel test is used to compare *a priori* groups. In this case, the second distance matrix is based on hypothesized data about the sample units (e.g., dummy variables coding for treatments). In fact, the generalized ANOSIM statistic from the last chapter is an example of this, but with ranked distances.

See Manly & Navarro Alberto (2017, section 5.6), McCune & Grace (2002, chapter 27) and Legendre & Legendre (2012, section 10.5.1) for more details and examples. Legendre & Legendre (2012) emphasize that Mantel tests should be used to test hypotheses about distances, not about the original variables.

Mantel tests can also be extended to examine the correlation between two matrices while controlling for the effect of a third. This is called a partial Mantel test, and is discussed by Legendre & Legendre (2012, section 10.5.2) and Goslee & Urban (2007). Be aware, however, that support for this is not universal: see Guillot & Rousett (2013) for details.

Legendre et al. (2015) argue that Mantel tests should not be used to detect or control for spatial correlation, and several recent articles have proposed new or alternative approaches to deal with this issue (Lisboa et al. 2014; Crabot et al. 2019).

van Mantgem & Schwilk (2009) used Mantel tests to test for spatial autocorrelation in fire effects. Recent ecological examples include Sun et al. (2019), who compared soil fungal assemblages beneath different shrubs, and Yu et al. (2018), who compared geographic and genetic distances among eight populations of a tropical tree.

Key Takeaways

A Mantel test is simply the correlation between two distance matrices obtained from the same sample units. These are often either based on two categories of data or one category of data and a matrix coding an experimental design.

The test statistic ranges from -1 (the two sets of distances are negatively correlated) to +1 (the two sets of distances are positively correlated).

Basic Procedure

The basic procedure for a Mantel test is as follows.

1. Convert each data matrix to a dissimilarity matrix. Use an appropriate distance measure for each matrix; the measures do not have to be the same.
2. Calculate the correlation between the two dissimilarity matrices. This is the test statistic. As a correlation, this always ranges between -1 and +1.
3. Assess statistical significance via a permutation test:
 - Shuffle the row and column identities of one dissimilarity matrix, permuting row and column identities together so that matrix symmetry is preserved. For example, if row A is moved down four positions, column A is also moved to the right four positions. Equivalently (and easier to describe), permute the rows of one of the original data matrices

and then recalculate the dissimilarity matrix.

- Recalculate the correlation between the two dissimilarity matrices. Save this value.
- Re-shuffle and recalculate the specified number of times. The permutations produce a sampling distribution of correlation values against which the actual test statistic is compared.

4. Calculate the P -value as the proportion of permutations that yielded equal or stronger correlations than the actual data did. Note that Mantel tests generally result in a one-tailed test, as in this case. A two-tailed test would be used if we had no *a priori* suspicion about whether the correlation would be positive or negative. To obtain the P -value for a two-tailed test, we would calculate the proportion of permutations whose correlation coefficient (in absolute value) was greater than the test statistic.

Simple Example, Worked By Hand

We'll start with the dissimilarity matrix based on the two response variables:

```
Resp.dist <- dist(perm.eg[, c("Resp1", "Resp2")])
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	2.828				
Plot3	4.123	2.236			
Plot4	11.314	11.662	9.849		
Plot5	9.849	9.220	7.071	4.123	
Plot6	12.207	12.042	10.000	2.236	3.162

For clarity, distances between plots from different groups are shown in bold.

We also need to calculate a distance matrix based on our grouping variable (A, B). This variable is of class 'character', so we need to change it to class numeric before we can calculate a distance matrix from it. Somewhat confusingly, you need to convert it to a factor before you can express the levels of the factor numerically. Euclidean distances are used to express differences among levels in a factor.

```
Group.dist <- as.numeric(as.factor(perm.eg$Group)) |>
dist()
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	0				
Plot3	0	0			
Plot4	1	1	1		
Plot5	1	1	1	0	
Plot6	1	1	1	0	0

The grouping variable is now expressed as a '0' if the plots are in the same group and a '1' if they are in different groups. For clarity, distances between plots from different groups are also shown in bold.

We could examine the correlation between these two distance matrices directly, but it's easier to track the data if we first combine them into a single object. To convert each distance matrix to a vector, we need to change it to class numeric. We will save these to a new dataframe, naming each column as we create it.

```
mantel.eg <- data.frame(
  Resp = Resp.dist,
  Group = Group.dist
)
```

Look at this object to ensure that you understand it.

The test statistic is the simple correlation between the two columns of values:
`with(mantel.eg, cor(Resp, Group))`

```
[1] 0.9390683
```

As with other tests, the significance of this statistic is assessed with a permutation test (not shown here).

Implementation in R (`vegan::mantel()`)

You could easily implement the steps in a Mantel test using functions such as `cor()`, `sample()`, etc. Of course, you could also use an existing function. For example, `mantel()` functions are available in the `vegan` and `ecodist` packages. Note that these two functions have the same name, which can cause problems if both packages are open simultaneously.

We'll use the `mantel()` function in the `vegan` package. The usage of this function is:

```
mantel(
  xdis,
  ydis,
  method = "pearson",
  permutations = 999,
  strata = NULL,
  na.rm = FALSE,
  parallel = getOption("mc.cores")
)
```

The arguments are:

- `xdis` – the first of two dissimilarity matrices to be tested
- `ydis` – the second of two dissimilarity matrices to be tested
- `method` – the correlation method ("pearson", "spearman", or "kendall") to be applied. The Pearson product-moment correlation is the default. Spearman and Kendall are non-parametric correlations based on the ranks of the distances.
- `permutations` – number of permutations to conduct to assess the significance of the correlation, or a list of permutation instructions obtained using the `how()` function. This latter option is necessary with complex designs where permutations need to be restricted – we'll discuss this **in a few classes**. Default is to conduct 999 permutations without any restrictions to the permutations.
- `strata` – integer vector or factor identifying strata that permutations are to be restricted within.
- `na.rm` – remove missing observations? Default is no (**FALSE**)
- `parallel` – option for parallel processing.

The results of the Mantel test are saved in an object of class 'mantel'. This class of object does not currently have `summary()` or `plot()` functions associated with it. However, you can explore the object in more detail using the `str()` function.

The `vegan` package also includes a `mantel.partial()` function.

Simple Example

Here, we'll call the two distance matrices that we created above:

```
simple.results.mantel <- mantel(
  xdis = Resp.dist,
  ydis = Group.dist,
  method = "pearson",
  permutations = 999
)

simple.results.mantel
```

```
Mantel statistic based on Pearson's product-moment correlation
Call:
mantel(xdis = Resp.dist, ydis = Group.dist, method = "pearson",
permutations = 999)
Mantel statistic r: 0.9391
Significance: 0.1
Upper quantiles of permutations (null model):
  90%   95%  97.5%   99%
0.0283 0.9391 0.9391 0.9391
Permutation: free
Number of permutations: 719
```

Grazing Example

The script for loading the oak data included calculation of the Bray-Curtis distance matrix based on the compositional data. To conduct a Mantel test, we also need a distance matrix expressing the differences in grazing status:

```
graz.dist <- as.numeric(as.factor(grazing)) |>
  dist()
```

Now we can compare the two distance matrices:

```
grazing.results.mantel <- mantel(
  xdis = Oak1.dist,
  ydis = graz.dist,
  method = "pearson",
  permutations = 999
)
```

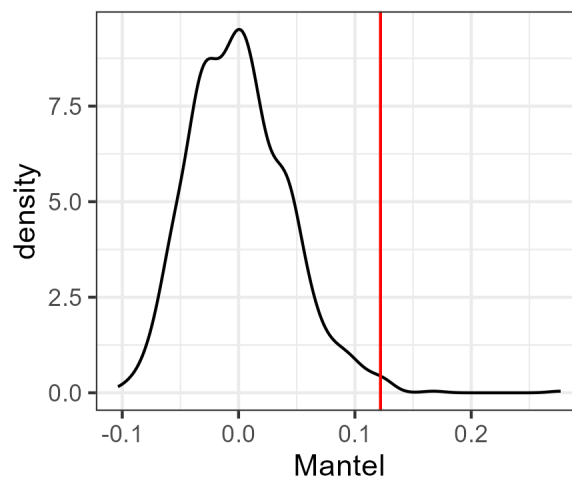
```
grazing.results.mantel
```

We included the default correlation method and number of permutations for completeness. If we were ok with the defaults, we could omit those arguments.

```
Mantel statistic based on Pearson's product-moment correlation
Call:
mantel(xdis = Oak1.dist, ydis = graz.dist, method = "pearson",
permutations = 999)
Mantel statistic r: 0.122
Significance: 0.009
Upper quantiles of permutations (null model):
  90%   95%  97.5%   99%
0.0555 0.0815 0.1014 0.1144
Permutation: free
Number of permutations: 999
```

Note, as for all other permutation tests, that the P -value differs slightly among individual runs of this test due to differences among permutations. However, the correlation itself is not affected by the number or identity of the permutations.

Using the `str()` function, can you figure out how I created this graph of the actual correlation and the distribution of correlations obtained from permutations?



Density distribution of the Mantel test statistic when testing for differences in composition between stands that were and were not currently grazed at the time of data collection. The red line is the actual value of the statistic. The density distribution was produced from 1000 permutations.

References

Crabot, J., S. Clappe, S. Dray, and T. Datry. 2019. Testing the Mantel statistic with a spatially-constrained permutation procedure. *Methods in Ecology and Evolution* 10:532-540.

- Goslee, S.C., and D.L. Urban. 2007. The ecodist package for dissimilarity-based analysis of ecological data. *Journal of Statistical Software* 22(7):1-19.
- Guillot, G., and F. Rousett. 2013. Dismantling the Mantel tests. *Methods in Ecology and Evolution* 4:336-344.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. 3rd English edition. Elsevier, Amsterdam, The Netherlands.
- Legendre, P., M.-J. Fortin, and D. Borcard. 2015. Should the Mantel test be used in spatial analysis? *Methods in Ecology and Evolution* 6:1239-1247.
- Lisboa, F.J.G., P.R. Peres-Neto, G.M. Chaer, E.d.C. Jesus, R.J. Mitchell, S.J. Chapman, and R.L.L. Berbara. 2014. Much beyond Mantel: bringing Procrustes Association Metric to the plant and soil ecologist's toolbox. *PLoS ONE* 9(6): e101238. doi:10.1371/journal.pone.0101238
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- Mantel, N. 1967. The detection of disease clustering and a generalized regression approach. *Cancer Research* 27:209-220.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Sun, Y., Y. Zhang, W. Feng, S. Qin, and Z. Liu. 2019. Revegetated shrub species recruit different soil fungal assemblages in a desert ecosystem. *Plant and Soil* 435:81-93.
- van Mantgem, P.J., and D.W. Schwilk. 2009. Negligible influence of spatial autocorrelation in the assessment of fire effects in a mixed conifer forest. *Fire Ecology* 5:116-125.
- Yu, N., J. Yuan, G. Yin, J. Yang, R. Li, and W. Zou. 2018. Genetic diversity and structure among natural populations of *Mytilaria laosensis* (Hamamelidaceae) revealed by microsatellite markers. *Silvae Genetica* 67:93-98.

Media Attributions

- Mantel

19. MRPP

Learning Objectives

- To understand the theory behind MRPP.
- To apply Mantel tests manually and through R.

Key Packages

```
require(vegan)
```

Theory

Multi-Response Permutation Procedure (MRPP) tests whether there is a significant difference between two or more groups of sampling units. It was first used to analyze the spatial distribution of archeological artifacts (Berry et al. 1980, 1983) and applied to ecological questions by Biondini et al. (1985 (Appendix), 1988). More recent ecological examples include Altrichter et al. (2018), Bates & Davies (2018), and Phillips & Swanson (2018).

MRPP focuses on the distances among sample units within each group. The test statistic (delta: δ) is the mean distance among sample unit within groups. **The key idea is simple: if groups differ, the mean within-group dissimilarity should be smaller than the mean dissimilarity among randomly selected groups of the same size.**

Generally, the mean distances are weighted by sample size, so that groups that contain more information (i.e., more sample units) are given more importance in the calculation. McCune & Grace (2002, Table 24.1) outline several other potential weighting methods. In my experience, these methods are not commonly used.

A blocked MRPP (MRBP) is also available, which allows you to test the significance of one factor while controlling for the effect of another. However, MRPP is a simple metric and is not appropriate for complex designs.

Key Takeaways

MRPP tests whether the mean within-group dissimilarity is smaller than expected.

The test statistic, δ , is simply the mean dissimilarity between pairs of sample units from the same group. It can be 0 or positive. Statistical significance focuses on the left tail of the distribution obtained from permutations of the grouping factor.

An effect size, A , is also calculated and relates the observed δ to its expected values if the grouping factor was not helpful.

MRPP cannot accommodate complex designs.

Basic Procedure

The basic procedure for MRPP is as follows.

1. Convert the data matrix to a dissimilarity matrix.
2. Calculate the mean distance within each group.
3. Calculate the test statistic (the observed δ) as the average of the mean distance within each group, weighting by sample size if necessary.
4. Assess statistical significance via a permutation test:
 - Shuffle the group identities.
 - Recalculate δ . Save this value.
 - Re-shuffle and recalculate the specified number of times. The permutations produce a sampling distribution of δ .
5. Calculate the P -value as the proportion of permutations that yielded δ equal or smaller than the observed δ (do you understand why we are interested in smaller values?).

MRPP also calculates an effect size, A , which is defined as the chance-corrected within-group agreement. A is calculated as:

$$A = 1 - \frac{\delta}{m_{\delta}} = 1 - \frac{\text{observed}\delta}{\text{expected}\delta}$$

where m_{δ} is the random expectation (i.e., value of δ if plots were randomly assigned groups). A ranges from 0 (groups contain as much heterogeneity as would be expected by chance) to 1 (all items are identical within groups). McCune & Grace (2002, p. 191) note that small A values (< 0.1) are common in community ecology.

Simple Example, Worked By Hand

Here's the distance matrix from our simple example:

```
Resp.dist <- dist(perm.eg[, c("Resp1", "Resp2")])
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	2.828				
Plot3	4.123	2.236			
Plot4	11.314	11.662	9.849		
Plot5	9.849	9.220	7.071	4.123	
Plot6	12.207	12.042	10.000	2.236	3.162

Recall that plots 1-3 are in group A and plots 4-6 are in group B. The distances between plots in the same group are in bold. This is the opposite of how I summarized it for other tests, but I do so here because MRPP relies only on the within-group dissimilarities for the calculation of its test statistic.

Calculation of the mean within-group distances is simple arithmetic:

- Group A: $(2.828 + 4.123 + 2.236)/3 = 3.063$
- Group B: $(4.123 + 2.236 + 3.162)/3 = 3.174$

Since our dataset is balanced, both groups receive equal weight and the average within-group distance across groups is $(3.063 + 3.174)/2 = 3.118$. This is the **observed delta**.

The expected delta is approximately equal to the mean distance averaged across the entire distance matrix – in this case, 7.461 (can you calculate this?). Therefore, the chance-corrected within-group agreement (A) is approximately:

$$A = 1 - \frac{3.118}{7.461} = 0.582$$

During the actual implementation of this analysis, the expected delta is calculated as the mean of the δ values calculated during the permutations.

Implementation in R (`vegan::mrpp()`)

In R, MRPP can be conducted using the `mrpp()` function in the `vegan` package. Its usage is:

```
mrpp(  
  dat,  
  grouping,  
  permutations = 999,  
  distance = "euclidean",  
  weight.type = 1,  
  strata = NULL,
```



```
parallel = getOption("mc.cores")
)
```

The arguments are:

- `dat` – data matrix, data frame, or dissimilarity matrix
- `grouping` – factor or numeric index for grouping observations
- `permutations` – number of permutations to conduct to assess the significance of the test statistic, or a list of permutation instructions obtained using the `how()` function. This latter option is necessary with complex designs where permutations need to be restricted – we'll discuss this in a few days. Default is to conduct 999 permutations without any restrictions to the permutations.
- `distance` – Default is Euclidean, though any distance measure available in `vegdist()` can be specified. This is only used if `dat` is not a dissimilarity matrix. Note that `mrpp()` decides whether it needs to calculate the dissimilarity matrix based on the class of the object identified with the `dat` argument. If the object is not of class 'dist', `mrpp()` calls `vegdist()` and converts the object to a dissimilarity matrix using the distance measure specified with the `distance` argument.
- `weight.type` – This is equivalent to the methods for weighting groups as outlined in Table 24.1 of McCune & Grace (2002). The default is method 1, which weights each group by its sample size.
- `strata` – integer vector or factor identifying strata that permutations are to be restricted within. Used for blocked MRPP (MRBP). I'm not sure whether this involves median alignment within blocks as discussed by McCune & Grace (2002, p. 194).
- `parallel` – option for parallel processing

The resulting object include several statistics:

- `A` – chance corrected within-group agreement.
- `delta` – observed delta; weighted mean within-group distance from actual data.
- `E.delta` – expected delta; weighted mean distance from permutations.
- `Pvalue` – significance of delt; number of permutations with delta as small or smaller than delta.

Simple Example, in R

To analyze this simple example:

```
simple.results.mrpp <- mrpp(
  dat = Resp.dist,
  grouping = perm.eg$Group,
  permutations = 999
)
```

```
simple.results.mrpp
```

```
Call:
mrpp(dat = Resp.dist, grouping = perm.eg$Group, permutations = 999)

Dissimilarity index: euclidean
Weights for groups:  n
Class means and counts:
```

```

      A      B
delta 3.063 3.174
n      3      3

Chance corrected within-group agreement A: 0.5821
Based on observed delta 3.118 and expected delta 7.461

Significance of delta: 0.1
Permutation: free
Number of permutations: 719

```

Grazing Example, in R

To analyze our grazing example:

```

grazing.results.mrpp <- mrpp(
  dat = Oak1.dist,
  grouping = grazing,
  permutations = 999
)

```

grazing.results.mrpp

```

Call:
mrpp(dat = Oak1.dist, grouping = grazing, permutations = 999)
Dissimilarity index:
bray

Weights for groups:  n
Class means and counts:      No      Yes
delta 0.6892 0.6877
n      30      17
Chance corrected within-group agreement A: 0.0183
Based on observed delta 0.6887 and expected delta 0.7015
Significance of delta: 0.001
Permutation: free
Number of permutations: 999

```

Although grazing is statistically significant, the chance-corrected within-group agreement (A) is very low, suggesting that this is not a strong effect. We may want to consider, for example, whether it is biologically meaningful.

As we saw with ANOSIM and Mantel tests, the object containing the results of this analysis includes more information than is displayed on the screen.

```
str(grazing.results.mrpp)
```

For example, you can plot the distribution of deltas obtained from the permutations and see how the observed δ compares.

References

- Altrichter, K.M., E.S. DeKeyser, B. Kobiela, and C.L.M. Hargiss. 2018. A comparison of five wetland communities in a North Dakota fen complex. *Natural Areas Journal* 38:275-286.
- Bates, J.D., and K.W. Davies. 2018. Characteristics of intact Wyoming big sagebrush associations in southeastern Oregon. *Rangeland Ecology and Management* 72:36-46.
- Berry, K.J., K.L. Kvamme, and P.W. Mielke Jr. 1980. A permutation technique for the spatial analysis of the distribution of artifacts into classes. *American Antiquity* 45:55-59.
- Berry, K.J., K.L. Kvamme, and P.W. Mielke Jr. 1983. Improvements in the permutation test for the spatial analysis of the distribution of artifacts into classes. *American Antiquity* 48:547-553.
- Biondini, M.E., C.D. Bonham, and E.F. Redente. 1985. Secondary successional patterns in a sagebrush (*Artemisia tridentata*) community as they relate to soil disturbance and soil biological activity. *Vegetatio* 60:25-36.
- Biondini, M.E., P.W. Mielke Jr., and K.J. Berry. 1988. Data-dependent permutation techniques for the analysis of ecological data. *Vegetatio* 75:161-168.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Phillips, P., and B.J. Swanson. 2018. A genetic analysis of dragonfly population structure. *Ecology and Evolution* 8:7206-7215.

20. PERMANOVA

Learning Objectives

To explore PERMANOVA, a particularly powerful analytical technique for multivariate data.

To understand the flexibility and limitations of PERMANOVA.

Readings

Anderson (2001)

Key Packages

```
require(vegan)
```

Introduction

PERmutational Multivariate ANalysis of VAriance (PERMANOVA) is a permutation-based technique – it makes no distributional assumptions about multivariate normality or homogeneity of variances. It can be thought of visually or geometrically.

PERMANOVA is equivalent to MRPP under certain conditions (Reiss et al. 2009), but is more versatile and widely applicable. Understanding its versatility requires also considering its flexibility and its limitations.

Theory

Recall that ANOVA quantifies the variation within a dataset as the sum of squared differences between points and their mean or centroid. This is used to calculate the total variation (total sum

of squares; SST) based on the differences between points and the grand mean. It is also used to calculate the variation within groups (SSW) from the differences between points and their group means. SSW is some subset of SST; variation that is not due to SSW occurs between/among groups (SSB).

The key insight behind PERMANOVA is that **the variation within a group can be calculated directly from a distance matrix. Specifically, the sum of squared differences between points and their centroid is equal to the sum of the squared interpoint distances divided by the number of points.** This idea is illustrated below (Fig. 2 from Anderson 2001). It is known as Huygens' theorem as it was first formulated by Christiaan Huygens – in the 17th century (Anderson et al. 2008)!

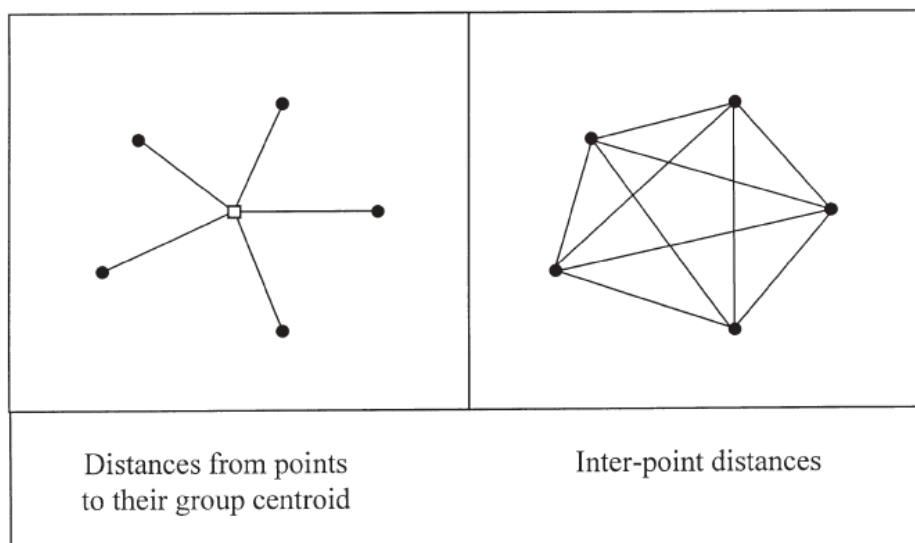


Fig. 2. The sum of squared distances from individual points to their centroid is equal to the sum of squared inter-point distances divided by the number of points.

Huygens' theorem means that variation can be calculated without knowing the location of group centroids. Anderson (2015) describes this as **geometric partitioning**. While this may seem abstract, it is crucial for semimetric distance measures (e.g., Bray-Curtis) that do not satisfy the triangle inequality (please review the Distance Measures chapter if this does not sound familiar). When using a semimetric distance measure, we cannot determine the locations of centroids from the locations of the data. By avoiding this step, **basic ANOVA theory can be applied to data summarized with any distance measure**. More details, including additional information about the mathematics of PERMANOVA, can be found in Anderson (2001, 2017), McArdle & Anderson (2001), and Anderson et al. (2008).

Knowing that we can calculate variation from the distance matrix means that we can also partition that variation among sources. The resulting technique requires no assumptions about normality and/or homogeneity of variances. **The only assumption is that observations are interchangeable under the null hypothesis** – the same assumption as with all permutation tests, and the justification for using permutations to assess significance.

The PERMANOVA test statistic, pseudo- F , is modeled directly after the conventional ANOVA F -statistic. It is calculated in the same way, as a ratio of the amount of variation between versus within groups, with the numerator and denominator each weighted by their degrees of freedom. It is 0 or positive, with larger values corresponding to larger proportional importance of the grouping factor. As with other permutational techniques, the distribution of the test statistic is assessed by

repeatedly relabelling observational units (i.e., permuting group identities) and recalculating the test statistic.

Since it is based on the distance matrix, PERMANOVA can be applied identically to both univariate and multivariate data. An important, and reassuring, point: the test statistic is **identical** to the conventional F -statistic when calculated using Euclidean distance for a single variable.

PERMANOVA can be used to analyze complex models, including multiple factors, covariates, and interactions between terms. More information about this is provided in the chapters about complex designs and restricting permutations.

Laughlin et al. (2004) provide an example of how these techniques can be applied to plant community data with repeated measurements (i.e., compositional data from permanent plots sampled in multiple years). Hudson & Henry (2009) used `adonis()` while Assis et al. (2018) used `adonis2()`; we discuss both of these functions below. Sharma et al. (2019) use both PERMANOVA (though they don't specify which function within `vegan`) and `anosim()` to examine the microbiome of people and their built environment.

Key Takeaways

PERMANOVA compares the variation between groups to the variation within groups.

The test statistic, pseudo- F , is modeled after the F -statistic from ANOVA. It is 0 or positive, with larger values corresponding to larger proportional importance of the grouping factor.

PERMANOVA is an extremely powerful and flexible technique. It can be applied to data of any dimensionality (including univariate) and expressed through any distance measure. It can also be applied to complex designs, including multiple factors and covariates (continuously distributed predictors).

Basic Procedure

The basic procedure for PERMANOVA is as follows.

1. Convert data matrix to a distance matrix, using an appropriate distance measure.
2. Square the distance matrix.
3. Calculate three partitions of the variation:
 - Total variation (total sum of squares; SST) – the sum of squared distances divided by the number of plots.
 - Variation within groups (sum of squares within groups; SSW) – calculated identical to SST but separately for each group. Add the value from each group together to yield the overall SSW .
 - Variation between groups (sum of squares between groups; SSB) – calculated by simple subtraction (i.e., $SSB = SST - SSW$).
4. Calculate the test statistic. The test statistic is termed a 'pseudo- F statistic' to distinguish it from the traditional parametric univariate F -statistic, but is calculated using the same formula:

$$PseudoF = \frac{SSB/(t - 1)}{SSW/(N - t)}$$

where SSB and SSW are as calculated above, t is the number of groups, and N is the total number of sample units.

5. Assess statistical significance via a permutation test:
 - Permute group identities.
 - Recalculate pseudo- F statistic
 - Repeat the specified number of times. The permutations produce a distribution of pseudo- F values against which the actual pseudo- F statistic value is compared.
6. Calculate the P -value as the proportion of permutations that yielded a pseudo- F value equal or greater than the actual data did.

Simple Example, Worked By Hand

Using the data and group identities from our simple example, let's work through the calculations involved with PERMANOVA.

We begin with our distance matrix. For clarity, the distances between plots from different groups are in bold (view `perm.eg$Group` to confirm this).

```
d.eg <- dist(perm.eg[, c("Resp1", "Resp2")])
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	2.828				
Plot3	4.123	2.236			
Plot4	11.314	11.662	9.849		
Plot5	9.849	9.220	7.071	4.123	
Plot6	12.207	12.042	10.000	2.236	3.162

Square the distance matrix to yield squared distances among plots.

```
d.eg_2 <- d.eg^2
```

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	8				
Plot3	17	5			
Plot4	128	136	97		
Plot5	97	85	50	17	
Plot6	149	145	100	5	10

Sum the squared distances and divide by the number of plots to obtain the total sum of squares (SST):

$$SST = \frac{1049}{6} = 174.83$$

```
SST <- sum(d.eg_2) / length(perm.eg$Group)
```

Note that this is Huygens' theorem in action!

We obviously could combine these steps. For example, here's a function to combine all of these steps:

```
SS.calculation <- function(x, variables) {  
  n <- nrow(x)  
  x %>%  
    dplyr::select(any_of(variables)) %>%  
    dist() %>%  
    `^` (2) %>%  
    sum() / n  
}
```

```
SS.calculation(x = perm.eg,  
  variables = c("Resp1", "Resp2"))
```

```
[1] 174.8333
```

Review the order of operations as outlined in the pipe. One unusual element here is the squaring of the distance matrix; for reasons that I don't understand I was able to do this using the magrittr pipe (%>%) but not the base R pipe (|>). This function is available in the GitHub repository.

Now let's calculate the variation within each group (i.e., SSW). We can manually identify the appropriate squared distances from the above matrix and do the calculation:

$$SSW = \frac{8 + 17 + 5}{3} + \frac{17 + 5 + 10}{3} = \frac{30}{3} + \frac{32}{3} = 20.67$$

Or, we can use `SS.calculation()` to calculate the variation within each group – all we need to do is index the object to refer to the group of interest:

```
SSW.A <- SS.calculation(x = perm.eg |> filter(Group == "A"),  
  variables = c("Resp1", "Resp2"))
```

```
SSW.B <- SS.calculation(x = perm.eg |> filter(Group == "B"),  
  variables = c("Resp1", "Resp2"))
```

```
SSW <- SSW.A + SSW.B
```

Calculate the variation between groups (*SSB*) as the difference between *SST* and *SSW*:

$$SSB = SST - SSW = 174.83 - 20.67 = 154.16$$

```
SSB <- SST - SSW
```

Finally, calculate the pseudo-*F* statistic:

$$PseudoF = \frac{SSB/(t-1)}{SSW/(N-t)} = \frac{154.16/1}{20.67/4} = 29.83$$

```
pseudo.F <- (SSB / (2 - 1)) / (SSW / (6 - 2))
```

Often, we will summarize this information in an ANOVA table:

Source	df	SS	MS	Pseudo-F
Group	1	154.16	154.16	29.83
Residual	4	20.67	5.17	
Total	5	174.83		

This table contains all of the data that we usually see in an ANOVA output except the P -value. Statistical significance will be determined by permuting the group identities, recalculating the pseudo- F statistic for each permutation, and comparing the observed value against the distribution of values obtained via permutation.

Can you figure out what the pseudo- F statistic is for the permutation of group identities {A,B,A,A,B,B}? As a hint, the squared distance matrix is repeated here, with bold text to indicate distances between plots from different groups.

	Plot1	Plot2	Plot3	Plot4	Plot5
Plot2	8				
Plot3	17	5			
Plot4	128	136	97		
Plot5	97	85	50	17	
Plot6	149	145	100	5	10

Implementation in R (`vegan::adonis2()`)

PERMANOVA is becoming increasingly common in community ecology. It is surprising, therefore, that only a few R packages provide it. We are using the main one, `vegan`.

The `LDM` (linear decomposition model) package provides another formulation of PERMANOVA. The authors claim that their formulation outperforms those in `vegan`. It was developed for use with microbiome data and seeks to simultaneously test global hypotheses (i.e., overall assessment of compositional differences among treatments) and individual operational taxonomic units (OTUs; i.e., response variables) (Hu & Satten 2020). Finally, the `PERMANOVA` package was released in 2021 and updated in late 2022 (Vicente-Gonzalez & Vicente-Villardón 2022). I have not yet evaluated either of these packages.

Several other packages also offer permutational multivariate analysis of variance and appear to be independent, but require the `vegan` package so often (I've not verified all) are using `vegan`'s functions. These include:

- **GUniFrac** – this package provides Generalized UniFrac distances for comparing microbial communities (Chen et al. 2012). The package description states that it provides three extensions to PERMANOVA: a different permutation scheme (Freedman-Lane), an omnibus test using multiple matrices, and an analytical approach for approximating the PERMANOVA p-value. The PERMANOVA function here is `PermanovaG()`.
- **RVAideMemoire** – this package provides miscellaneous functions useful in biostatistics. The PERMANOVA functions here are `perm.anova()` and `adonis.II()`. According to the help files:
 - `perm.anova()` can be applied to 1 to 3 factors. For 2 or 3 factors, the experimental design must be balanced. For 2 factors, the factors can be crossed with or without interaction, or nested. The second factor can be a blocking (random) factor. For 3 factors, design is restricted to 2 fixed factors crossed (with or without interaction) inside blocks (third factor).
 - `adonis.II()` is a wrapper to `vegan::adonis()` but performs type II tests (whereas `vegan::adonis()` performs type I tests).
- **biodiversityR** – this package provides a graphical user interface (GUI) for community analysis – this can be useful in the moment but is difficult to script. Its website includes a downloadable book (Kindt & Coe 2005) about the analysis of community data, focusing specifically on trees. The PERMANOVA function here is `nested.npmanova()`.
- **ade4::amova?**
- **Capscale; anova.cca of dbrda**
- The **microbiomeSeq** package combines PERMANOVA with an ordination, so the statistical test is automatically accompanied by a visualization of the data. However, its website (http://userweb.eng.gla.ac.uk/umer.ijaz/projects/microbiomeSeq_Tutorial.html) is from 2017 yet has a disclaimer that it is still in development. I haven't explored the function it uses to conduct the PERMANOVA.
- **smartsnp** – `smart_permanova()` is based on `vegan::adonis()`.

PERMANOVA can be conducted using the `adonis2()` and `adonis()` functions in **vegan**. These two functions differ in scope, speed – the help file notes that `adonis2()` can be much slower than `adonis()` – and output. Confusingly, these functions yield objects of different classes. As of version 2.6-4, the **vegan** maintainers note that `adonis()` will be eliminated soon, so I do not describe it below. Also, some of the arguments described below for `adonis2()` are not available in older versions of this package.

vegan::adonis2()

As indicated by its name, `adonis2()` is a replacement for `adonis()`. Its usage is:

```
adonis2(
  formula,
  data,
  permutations = 999,
  method = "bray",
  sqrt.dist = FALSE,
  add = FALSE,
  by = "terms",
  parallel = getOption("mc.cores"),
  na.action = na.fail,
  strata = NULL,
  ...
)
```

The main arguments are:

- **formula** – model formula such as $Y \sim A + B * C$. See the 'Complex Models' chapter for more details of how to specify a model formula. For now, the key point is that we are specifying that we want to analyze the response (Y) as a function of (\sim) one or more explanatory variables (A , B , C). The class of Y determines what is done:
 - If Y is a dissimilarity matrix (i.e., output from `dist()` or `vegdist()`), the pseudo- F statistic is calculated directly.
 - If Y is a data frame or matrix containing data, `vegdist()` is applied to calculate the distance matrix first and then the pseudo- F statistic is calculated.
- **data** – the data frame in which the explanatory variables are located. Sample units are assumed to be in the same order in `data` as in the rows of Y .
- **permutations** – number of permutations to conduct to assess the significance of the pseudo- F statistic, or a list of permutation instructions obtained using the `how()` function. This latter option is particularly important for PERMANOVA, as it is necessary when permutations need to be restricted – see that chapter for details. Default is to conduct 999 permutations without any restrictions to the permutations.
- **method** – method of calculating the distance matrix, if needed (i.e., if Y in the formula is not a distance matrix). The `vegdist()` function is used to do these calculations, which is why the default is Bray-Curtis, though you can specify any distance method available in `vegdist()`.
- **sqrt.dist** – take square root of dissimilarities. This is one way to make semimetric dissimilarities behave in a more 'Euclidean' manner. Default is to not 'euclidify' them in this fashion (`FALSE`).
- **add** – add a constant to the dissimilarities so that no eigenvalues are negative. There are two options here, which we'll discuss in the context of Principal Coordinates Analysis (PCoA). Default is to not add this value (`FALSE`).
- **by** – How to deal with other terms when assessing significance. See below for additional discussion of types of sums of squares. Four options are available:
 - **terms** – tests terms in sequential order, from first to last in formula. The default. This is the Type I or 'sequential' sums of squares that you may recall from basic statistics courses.
 - **margin** – tests the marginal effect of each term, or its effect after accounting for all other terms in the model. This is Type III or 'partial' sums of squares that you may recall from basic statistics courses. If applied to a model with an interaction term, this will return just the interaction term and ignore the main effects.
 - **onedf** – tests terms as a series of contrasts, each with one df.
 - **NULL** – tests all terms together rather than individually.
- **parallel** – option for parallel processing
- **na.action** – what to do if explanatory variables are missing. Default is to not continue (`na.fail`).
- **strata** – groups that permutations are to be restricted within. Can alternatively be specified through the `permutations()` function, which is discussed in this chapter about restricted permutations.

The results from `adonis2()` are saved to an object of class "anova.cca". This object is also recognized as belonging to classes "anova" and "data.frame". It currently includes the degrees of freedom (`Df`), sums of squares (`SumOfSqs`), partial R^2 (`R2`), pseudo- F statistic (`F`), and P -value for each term (`Pr(>F)`) along with the pseudo- F statistics from the permutations (`attr(x, "F.perm")`) and lots of details about how permutations were conducted.

The partial R^2 for each term is calculated as the SS for the term as a proportion of SST. The partial R^2 values may NOT sum to 1 if you are testing multiple factors using a marginal model (`by = "margin"`) – see the discussion of types of sums of squares in the 'Complex Models' chapter.

Simple Example

```
simple.result.adonis2 <- adonis2(  
  perm.eg[, c("Resp1", "Resp2")] ~ Group,  
  data = perm.eg,  
  method = "euc"  
)
```

Calling this object prints the ANOVA table in the console.

```
Permutation test for adonis under reduced model  
Terms added sequentially (first to last)  
Permutation: free  
Number of permutations: 719  
  
adonis2(formula = perm.eg[, c("Resp1", "Resp2")] ~ Group, data = perm.eg,  
  method = "euc")  
      Df SumOfSqs      R2      F Pr(>F)  
Group   1  154.167 0.88179 29.839 0.1  
Residual 4   20.667 0.11821  
Total   5  174.833 1.00000
```

Verify the calculations we made by hand above. The ANOVA table is identical to what we calculated, though the order of the information is slightly different. Also, this table includes two elements that we did not calculate:

- The partial R^2 value for each term. As noted above, this is calculated as the SS for that term as a proportion of the total SS for the dataset. For example, the partial R^2 for Group = $154.167 / 174.833 = 0.88179$
- A permutation-based P -value for each term being tested (i.e., other than Residual and Total). Although Group has a large pseudo-F value, it is not statistically significant. This is a reflection of the limited number of permutations for a small sample size.

Grazing Example

```
grazing.result.adonis2 <- adonis2(  
  Oak1 ~ grazing,  
  method = "bray"  
)
```

```
Permutation test for adonis under reduced model  
Terms added sequentially (first to last)  
Permutation: free  
Number of permutations: 999  
  
adonis2(formula = Oak1 ~ grazing, method = "bray")  
      Df SumOfSqs      R2      F Pr(>F)  
grazing   1   0.6491 0.05598 2.6684 0.001 ***  
Residual 45  10.9458 0.94402  
Total    46  11.5949 1.00000  
---  
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Grazing clearly affects community composition, though it accounts for a relatively small amount of the variation in composition (5.6%).

We did not specify the `data` argument because grazing was saved as a separate object. If we wanted to, we could have called it by its column name within `oak` (or `oak_explan`). Also, if we had specified the distance matrix instead of the data matrix for `Y`, the `method` argument would have been unnecessary:

```
adonis2(Oak1.dist ~ grazing)
```

Extensions of PERMANOVA

I've introduced PERMANOVA as a means of comparing groups, but it is more broadly described as a **linear model**. Recall that ANOVA is a specific instance of a linear model in which the explanatory variable is categorical. Linear models can be used to compare groups (categorical variables) and to regress one continuously distributed variable on another. As such, PERMANOVA can be used to analyze any linear model. This is sometimes referred to more generally as a 'distance-based linear model' (DISTLM) or as distance-based redundancy analysis (dbRDA) (Legendre & Anderson 1999; McArdle & Anderson 2001).

The `adonis2()` function can handle both ANOVA and regression models. Each variable is treated as categorical or continuous based on its class. Therefore, it is helpful to always check that the degrees of freedom in a model output correspond to what you expected. A variable that is categorical will have 1 fewer df than it has levels, whereas a variable that is continuous will only have 1 df. More information is available in the 'Complex Models' chapter.

Limitations of `vegan::adonis2()`

Although PERMANOVA is an extremely powerful technique, its implementation in R has some limitations. Topics like the choice of sums of squares mentioned above are not unique to `adonis2()`. Two other issues to be aware of are how to handle random effects and how to decide what to permute.

Random Effects

Currently, `adonis2()` does not permit the specification of random effects. An alternative way to account for random effects is to use sequential sums of squares and fit the term that would otherwise be designated as a random effect as the very first term in the model. Doing so results in a model in which as much variation as possible is explained with that term before the other factor(s) are evaluated. However, using this approach is 'expensive' in terms of df.

For example, if plots were arranged in blocks (e.g., the whole plots of a split-plot design), block ID could be included as the first term in the model so that the variation among blocks is accounted for before testing other terms. Permutations would also have to be restricted to reflect this design.

Choice of What to Permute

In our earlier examples of permutations, we permuted the group assignment of each sample unit. We could alternatively have kept the group assignments unchanged but permuted the rows of the response matrix. Furthermore, the raw data can be permuted or the residuals obtained from a model can be permuted. In `adonis2()`, we are not able to specify which of these methods to use. However, this is likely a relatively minor issue: Anderson et al. (2008) note that these methods give very similar results.

Non-R Formulations of PERMANOVA

If the limitations of `adonis2()` are problematic for your analysis, you may want to explore a non-R formulation of PERMANOVA.

The PERMANOVA+ extension to PRIMER (<http://www.primer-e.com/>) is an extremely powerful and versatile way to use PERMANOVA. It was coded by Dr. Anderson (author of the 2001 article that developed this technique and introduced it to ecological audiences) and includes an extensive help manual (Anderson et al. 2008). Among other features, it permits you to choose which type of SS you would like to use and to analyze unbalanced data, random effects, etc. The basic approach requires that you create a design file containing the necessary information for partitioning the data based on the factors and experimental design, and then run the PERMANOVA on the distance matrix, choosing that design file during the analysis.

In PC-ORD (<https://www.pcord.com/>), PERMANOVA can be used to analyze fairly simple designs. Factors are identified in a second matrix (i.e., where explanatory variables are identified). Any of the distance measures available through PC-ORD can be used. Pairwise comparisons can be made to evaluate which groups differ in factors with more than two levels. However, the formulation of PERMANOVA in PC-ORD (version 6) has a number of limitations:

- Requires balanced data (same number of sample units in each group)
- Limited to one or two factors

Tang et al. (2016) presented PERMANOVA-S, an extension of PERMANOVA for microbiome data that i) can account for ‘confounders’ (confounding variables that are correlated with both the response matrix and explanatory variables), and ii) can compare conclusions across multiple distance measures simultaneously. For example, an analysis could compare the conclusions if based on Bray-Curtis distances (weighted by abundance) or on Jaccard distance (based on presence/absence data). A free program, in the C language, used to be available through Tang’s website (<https://tangzhengl.github.io/tanglab/software.html>) though as of January 2024 it no longer appears there.

Conclusions

PERMANOVA is an extremely powerful and versatile technique. It has strong parallels with ANOVA and thus is familiar to most ecologists.

The flexibility of PERMANOVA allows it to be used in complex models, but this means it can also be used incorrectly. See additional ideas in the chapters about complex models, controlling permutations, and restricting permutations.

References

- Anderson, M.J. 2001. A new method for non-parametric multivariate analysis of variance. *Austral Ecology* 26:32-46.
- Anderson, M.J. 2015. *Workshop on multivariate analysis of complex experimental designs using the PERMANOVA+ add-on to PRIMER v7*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK.
- Anderson, M.J. 2017. Permutational Multivariate Analysis of Variance (PERMANOVA). *Wiley StatsRef: Statistics Reference Online*. DOI: 10.1002/9781118445112.stat07841
- Anderson, M.J., and C.J.F. ter Braak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.
- Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK. 214 p.
- Assis, D.S., I.A. Dos Santos, F.N. Ramos, K.E. Barrios-Rojas, J.D. Majer, and E.F. Vilela. 2018. Agricultural matrices affect ground ant assemblage composition inside forest fragments. *PLoS One* 13(5):e0197697.
- Chen, J., K. Bittinger, E.S. Charlson, C. Hoffmann, J. Lewis, G.D. Wu, R.G. Collman, F.D. Bushman, and H. Li. 2012. Associating microbiome composition with environmental covariates using generalized UniFrac distances. *Bioinformatics* 28:2106-2113.
- Hu, Y-J., and G.A. Satten. 2020. Testing hypotheses about the microbiome using the linear decomposition model (LDM). *Bioinformatics* 36:4106-4115.
- Hudson, J.M.G., and G.H.R. Henry. 2009. Increased plant biomass in a high Arctic heath community from 1981 to 2008. *Ecology* 90:2657-2663.
- Kindt, R., and R. Coe. 2005. *Tree diversity analysis: a manual and software for common statistical methods for ecological and biodiversity studies*. World Agroforestry Centre (ICRAF), Nairobi, Kenya. <http://apps.worldagroforestry.org/downloads/Publications/PDFS/b13695.pdf>
- Laughlin, D.C., J.D. Bakker, M.T. Stoddard, M.L. Daniels, J.D. Springer, C.N. Gildar, A.M. Green, and W.W. Covington. 2004. Toward reference conditions: wildfire effects on flora in an old-growth ponderosa pine forest. *Forest Ecology and Management* 199:137-152.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McArdle, B.H., and M.J. Anderson. 2001. Fitting multivariate models to community data: a comment on distance-based redundancy analysis. *Ecology* 82:290-297.
- Sharma, A., M. Richardson, L. Cralle, C.E. Stamper, J.P. Maestre, K.A. Stearns-Yoder, T.T. Postolache, K.L. Bates, K.A. Kinney, L.A. Brenner, C.A. Lowry, J.A. Gilbert, and A.J. Hoisington. 2019. Longitudinal homogenization of the microbiome between both occupants and the built environment in a cohort of United States Air Force cadets. *Microbiome* 7:70.
- Tang, Z-Z., G. Chen, and A.V. Alekseyenko. 2016. PERMANOVA-S: association test for microbial community composition that accommodates confounders and multiple distances. *Bioinformatics* 32:2618-2625.
- Vicente-Gonzalez, L., and J.L. Vicente-Villardón. 2022. *PERMANOVA: Multivariate analysis of variance based on distances and permutations*. R Package, v.0.2.0. <https://cran.r-project.org/package=PERMANOVA>

Media Attributions

- Anderson.2001_Figure2

21. PERMDISP

Learning Objectives

To learn how to test for homogeneity of dispersion via PERMDISP.

Key Packages

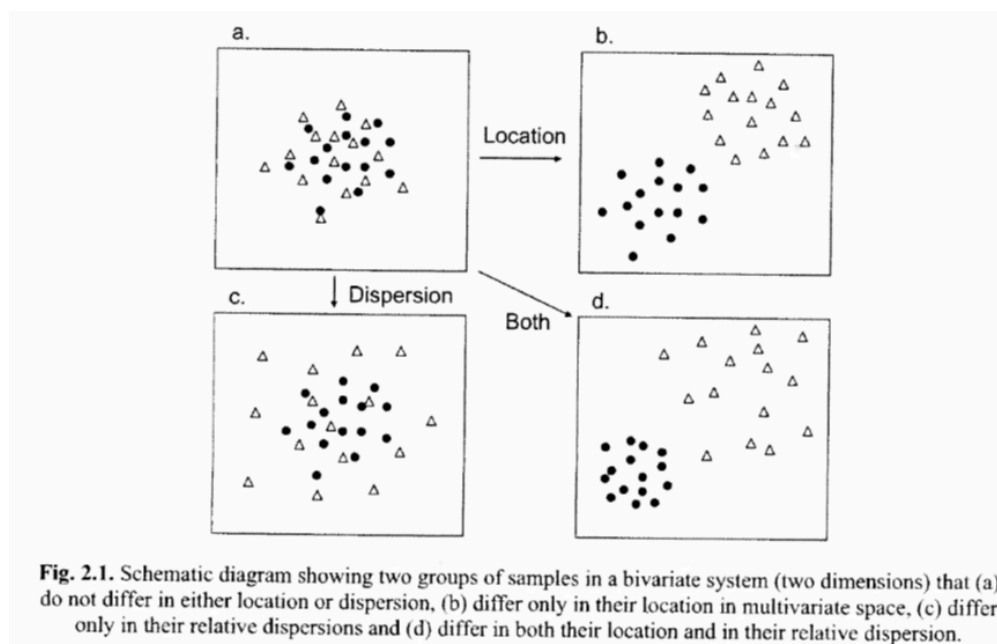
```
require(vegan)
```

Theory

Permutational techniques do not assume homogeneity of variances. However, patterns of dispersion (variability among sample units) can influence the interpretation of a statistical test. Significant differences among groups may occur for either or both of two reasons (Warton et al. 2012):

- Differences in the means (**centroids**)
- Differences in the amount of dispersion of sample units around those centroids

These alternatives are nicely illustrated in two dimensions – see Figure 2.1 from Anderson et al. (2008), reproduced below.



Sensitivity to variability is not unique to multivariate analyses – groups in a univariate analysis can also differ with regard to their mean values, variation around those means, or both. In univariate analyses, dispersion can be examined using Levene's test. PERMDISP is a multivariate extension of Levene's test (Anderson 2006) to examine whether groups differ in plot-to-plot variability.

In essence, PERMDISP involves calculating the distance from each data point to its group centroid and then testing whether those distances differ among the groups. However, recall that distance measures such as Bray-Curtis are semi-metric ... this means we cannot always calculate the centroid directly. In practice, therefore, a Principal Coordinates Analysis (PCoA) is applied to the distance measure and the centroids are calculated on the basis of the coordinates in PCoA space. See the 'PCoA' chapter for more details on this technique.

Consider an example in which a PERMANOVA test has indicated a difference among groups. Immediately, we know that the patterns are not as in Fig. 2.1a, but how do we determine whether the patterns are due to dispersion (Fig. 2.1c), location (Fig. 2.1b), or both (Fig. 2.1d)? A non-significant result from PERMDISP would indicate that groups do not differ in dispersion and that therefore the differences are entirely due to differences in location (e.g., Fig. 2.1b). A significant result from PERMDISP would indicate that the groups differ in dispersion. However, it is not possible on the basis of these two tests to determine conclusively whether the groups differ only in dispersion (Fig. 2.1c) or both dispersion and location (Fig. 2.1d). In research projects I've been involved with, we've visually explored our data to decide between these two options (this admittedly is not a very satisfying approach).

Anderson et al. (2008) include a chapter providing more detail about tests of homogeneity of dispersion.

Multivariate homogeneity – distance to group centroids – can also be used for other purposes:

- As a measure of beta diversity (Anderson et al. 2006)
- To estimate how the number of sampling units affects precision of compositional analyses (Anderson and Santana-Garcon 2015)

Anderson (2015) notes that dispersion is strongly affected by transformations applied to the original data and by the choice of distance measure.

If PERMDISP is being used as a follow-up to a PERMANOVA or other test, it is essential that the same data adjustments and distance measure be used in both cases, otherwise the tests are not directly comparable.

Basic Procedure

The basic procedure for PERMDISP is as follows:

1. Begin with a distance matrix. Conduct a Principal Coordinates Analysis (PCoA) ordination of the distance matrix. This expresses the sample units along new axes that are organized such that the first axis explains as much of the variation in the dataset as possible, the second axis explains as much of the remaining variation as possible, etc.
2. Overlay the grouping factor onto the PCoA. In other words, code each sample unit based on the group to which it belongs.
3. Calculate the centroid (average score) for each level of the grouping factor in the ordination space.
4. Calculate the distance between each observation and the centroid of the group to which it belongs.

This technique differs from the others that we've covered as it does not involve a statistical test. Rather, it produces a univariate dataset – the distance from each observation to the centroid of its group – that can be analyzed via other statistical tests. This response is likely to be relatively normally distributed regardless of the characteristics of the multivariate data that formed the distance matrix, and therefore could be analyzed via techniques like ANOVA. It of course could also be analyzed via PERMANOVA (`vegan::adonis2()`) or other permutation-based techniques.

Simple Example, Worked By Hand

I'll illustrate this approach by working directly from the data rather than the distance matrix. Our simple example is in Euclidean space so we can use techniques such as the Pythagorean formula to determine the group centroids. I'm also going to skip the PCoA ordination that is described below – it rotates the dataset but doesn't change the overall results when we work with Euclidean distances.

Begin by calculating the centroid for each group.

```
perm.eg.centroids <- perm.eg |>
  group_by(Group) |>
  summarize(Resp1.mean = mean(Resp1), Resp2.mean = mean(Resp2))
```

	Group	Resp1.mean	Resp2.mean
1	A	3	3
2	B	10	10.3

Combine the raw data with the group means, and use Pythagorean theorem to calculate the distance from each observation to the mean of its group.

```
perm.eg <- perm.eg |>
```

```
merge(y = perm.eg.centroids) |>
mutate(distance = sqrt((Resp1 - Resp1.mean)^2 + (Resp2 - Resp2.mean)^2))
```

	Group	Resp1	Resp2	Resp1.mean	Resp2.mean	distance
1	A	1	4	3	3.00000	2.236068
2	A	3	2	3	3.00000	1.000000
3	A	5	3	3	3.00000	2.000000
4	B	9	12	10	10.33333	1.943651
5	B	10	8	10	10.33333	2.333333
6	B	11	11	10	10.33333	1.201850

The column `perm.eg$distance` is the set of values that are produced by `PERMDISP`.

Implementation in R (`vegan::betadisper()`)

Analyses of multivariate homogeneity of group dispersions (variances) can be done using the `betadisper()` function in the `vegan` package. Its usage is:

```
betadisper(
  d,
  group,
  type = c("median", "centroid"),
  bias.adjust = FALSE,
  sqrt.dist = FALSE,
  add = FALSE
)
```

The main arguments are:

- `d` – distance matrix to be analyzed
- `group` – grouping factor
- `type` – type of analysis to perform: either adjust groups relative to their spatial median or to the group centroid. The spatial median is the default though the centroid is commonly used.
- The other arguments here provide adjustments for small sample size, negative eigenvalues, etc.

This function conducts a PCoA of the distance matrix (to express semi-metric distances in Euclidean space), calculates the distance from each sample unit to the centroid for its level of the grouping factor, and saves these distances (and other things) in an object of class ‘betadisper’. It does not test for differences in them.

Separate functions can be applied to an object of class ‘betadisper’ to accomplish different objectives:

- View data graphically
 - `plot()`
 - `boxplot()`
- Conduct statistical tests (note that these are univariate data!)
 - `anova()`
 - `TukeyHSD()`

◦ `permutest()`

Simple Example

The bivariate response in our simple example differs between the two groups. Do they differ in dispersion?

Use `betadisper()` to calculate the distance from each sample unit to the centroid of its group:

```
perm.eg.betadisper <- betadisper(d = dist(perm.eg[, c("Resp1", "Resp2")]), group =  
  perm.eg$Group, type = "centroid")
```

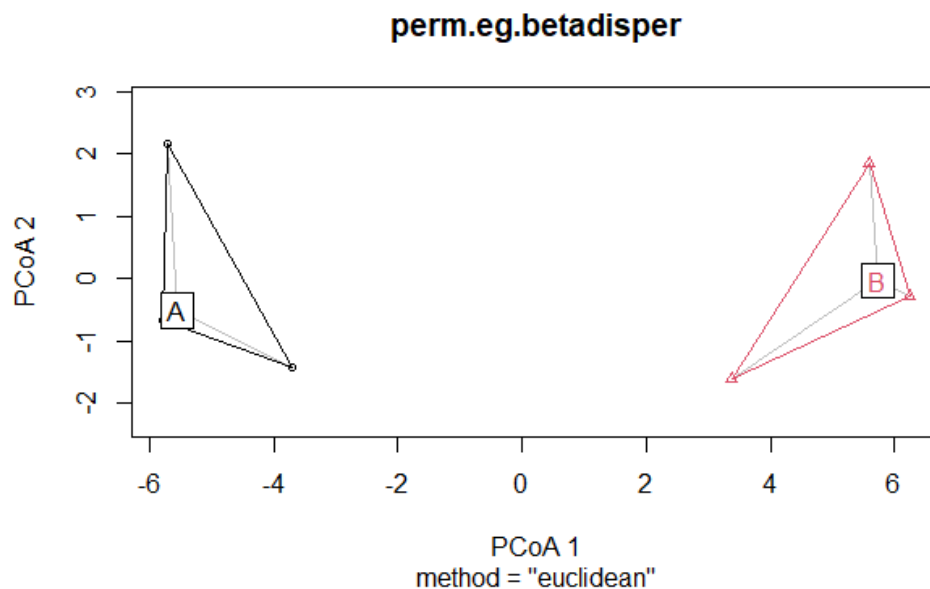
The data consist of the distance from each observation to the centroid of its group:

```
perm.eg.betadisper$distances
```

Confirm that these distances are identical to those that calculated by hand above.

We can graph the data:

```
plot(perm.eg.betadisper)
```



*PCoA ordination showing patterns between 6 sample units in two groups (A, B).
Sample unit positions are based on two response variables.*

Key features to note in this plot:

- The ordination is the first two axes of the Principal Coordinates Analysis (PCoA). With this bivariate dataset, the data cloud has simply been rotated so that the variation is aligned with

the ordination axes. The axes of the PCoA are combinations of the responses rather than individual responses. The responses were combined in such a way that the first axis explains as much of the variation in the dataset as possible, and the second axis explains the remaining variation in this bivariate response. We'll talk about this more when we discuss ordinations.

- Plots are shown as points, with colors and shapes identifying the group to which they belong.
- A colored line outlines the perimeter or hull of the group in this ordination space.
- Each group is labelled at its centroid.
- A grey line connects each stand to its centroid. These lines are the distances that were calculated in `betadisper()`.

We can whether the distances differ statistically:

```
anova(perm.eg.betadisper)
```

```
Analysis of Variance Table
Response: Distances
      Df Sum Sq Mean Sq F value Pr(>F)
Groups  1  0.0116  0.01163   0.0082  0.9321
Residuals 4  5.6555  1.41388
```

Of course, we could also do this as a permutation-based test. Let's use PERMANOVA to do so:

```
adonis2(dist(perm.eg.betadisper$distances) ~ perm.eg$Group)
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 719

adonis2(formula = dist(perm.eg.betadisper$distances) ~ perm.eg$Group)

Df SumOfSqs      R2      F Pr(>F)
perm.eg$Group  1  0.0116 0.00205 0.0082    0.9
Residual       4  5.6555 0.99795
Total         5  5.6672 1.00000
```

This test is not significant, so there is no evidence of a difference in dispersion between the two groups. This agrees with what we saw visually.

Grazing Example

We established previously that the composition of the oak plant community differs as a function of grazing history:

```
adonis2(Oak1.dist ~ grazing)
```

```
Permutation test for adonis under reduced model
```

```

Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1.dist ~ grazing)
      Df SumOfSqs      R2      F Pr(>F)
grazing  1   0.6491 0.05598 2.6684  0.001 ***
Residual 45  10.9458 0.94402
Total    46  11.5949 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Let's check whether these differences are due to location and/or dispersion. Summarize dispersion relative to the group centroids:

```
graz.res.betadisper <- betadisper(d = Oak1.dist, group = grazing, type = "centroid")
```

The data consist of the distance from each observation to the centroid of its group:

```
graz.res.betadisper$distances
```

Our simple example was bivariate so it was easy to envision the centroids. In this case the data are highly multivariate (based on 103 species). The centroids, and the associated distance from each stand to its centroid, are calculated based on all axes.

To test for differences via ANOVA:

```
anova(graz.res.betadisper)
```

```

Analysis of Variance Table

Response: Distances
      Df  Sum Sq  Mean Sq F value Pr(>F)
Groups  1 0.000463 0.0004627  0.0974 0.7564
Residuals 45 0.213788 0.0047508

```

The main effect of grazing is not significant, so there is no reason to test for pairwise differences between groups (and, we only have two groups anyways!). However, we will do so anyways to show how it could be used in other analyses:

```
TukeyHSD(graz.res.betadisper)
```

```

Tukey multiple comparisons of means
 95% family-wise confidence level

Fit: aov(formula = distances ~ group, data = df)

```

\$group		diff	lwr	upr	p adj
Yes-No	-0.00653026	-0.04867375	0.03561323	0.7564127	

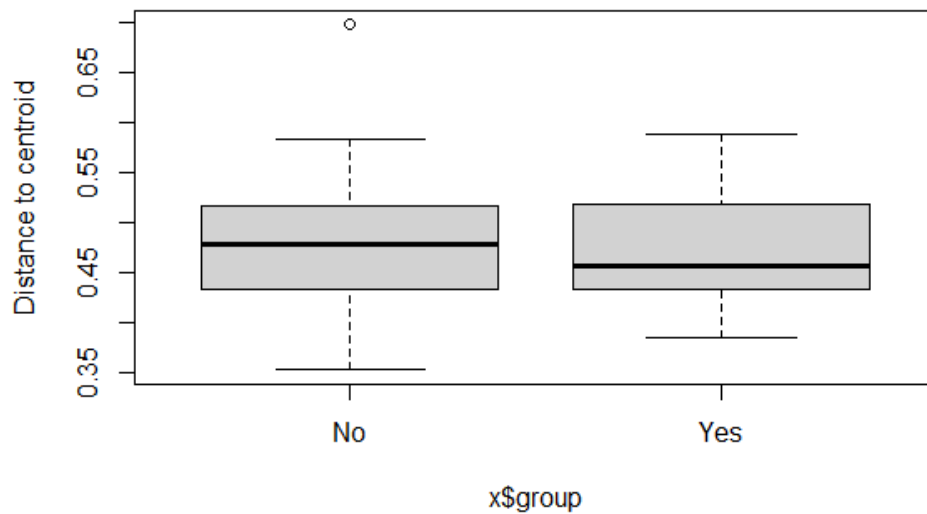
If our grouping factor included three or more groups, there would be one line and test for each pair of groups.

We conclude that, although composition differs among grazing statuses, it does so because of differences in location rather than dispersion.

Finally, let's explore these distances visually.

When applied to an object of class 'betadisper', the `boxplot()` function creates a box-and-whisker plot showing the distance to the group centroid in each group.

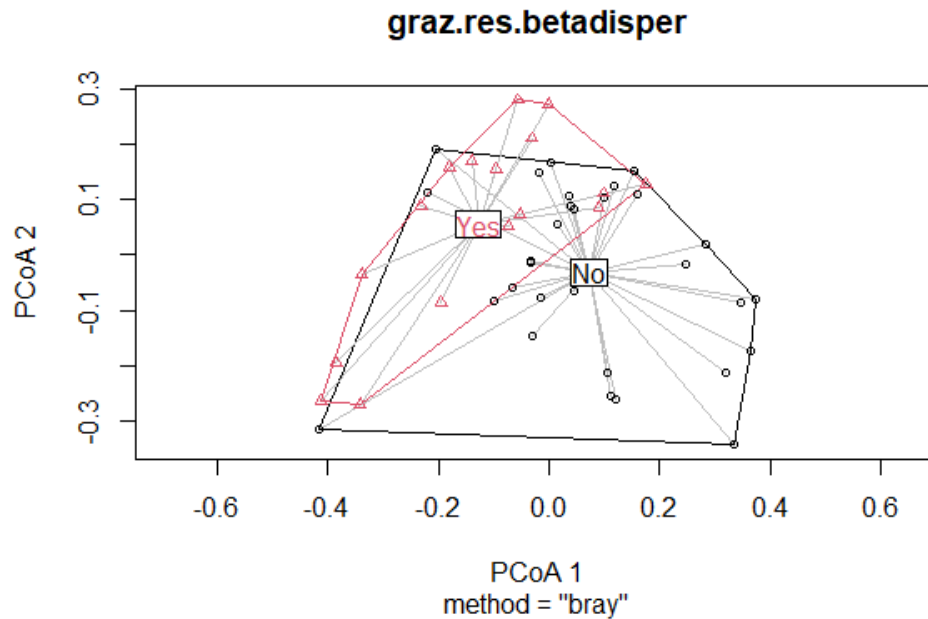
```
boxplot(graz.res.betadisper)
```



Box-and-whisker plot showing the distances from each stand to the centroid of the group that it belongs to. Distances were calculated from a PCoA of the Bray-Curtis dissimilarities between stands on the basis of the relativized compositional data of 103 common species.

And, as we've seen above, we can use the `plot()` function for an object of this class.

```
plot(graz.res.betadisper)
```

First two dimensions of a PCoA ordination of the oak plant community data. Each point is a stand. Symbol colors and shapes distinguish stands that were or were not experiencing grazing at the time of data collection. Grey lines connect each stand to the centroid of its group.

This plot is very similar to the one from the simple example but deserves a few additional comments:

- The underlying data are highly multivariate – 103 species – but we’re seeing just the first two axes of the Principal Coordinates Analysis (PCoA). The procedure for calculating these axes means that each axis explains as much of the variation as possible that has not been explained by other axes. In other words, there is no other two-dimensional representation of this dataset that reflects more of its variation than is possible with these two axes.
- A grey line connects each stand to its centroid in this 2-D ordination space. These lines are similar to but not exactly the same as the distances calculated in `betadisper()`, because the actual distances are based on the full PCoA (103 axes) rather than just the two axes shown here.

References

Anderson, M.J. 2006. Distance-based tests for homogeneity of multivariate dispersions. *Biometrics* 62(1):245–253.

Anderson, M.J. 2015. *Workshop on multivariate analysis of complex experimental designs using the PERMANOVA+ add-on to PRIMER v7*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK.

Anderson, M.J., K.E. Ellingsen, and B.H. McCordle. 2006. Multivariate dispersion as a measure of beta diversity. *Ecology Letters* 9:683–693.

Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK. 214 p.

Anderson, M.J., and J. Santana-Garcon. 2015. Measures of precision for dissimilarity-based multivariate analysis of ecological communities. *Ecology Letters* 18(1):66-73.

Warton, D.I., S.T. Wright, and Y. Wang. 2012. Distance-based multivariate analyses confound location and dispersion effects. *Methods in Ecology and Evolution* 3:89-101.

Media Attributions

- Anderson.et.al.2008_Figure2.1
- BETADISPER_perm.eg
- PERMDISP.boxplot
- PERMDISP.grazing

22. RRPP

Learning Objectives

To explore the utility of RRPP.

Key Readings

Recommended: Collyer & Adams (2018)

Key Packages

```
require(RRPP)
```

Introduction

Collyer & Adams (2018) recently introduced a new technique, ‘Randomization of Residuals in a Permutation Procedure’ (RRPP) that I think deserves attention. RRPP was developed for morphometric analysis but is applicable in many settings. I am still exploring this technique, but **it is particularly relevant for datasets where Euclidean distance measures are appropriate**. Telemeco & Gangloff (2020) present a readable description of how to use this technique to analyze a multivariate dataset consisting of a range of measures of how individual organisms respond to stress – see in particular the detailed tutorial in their Appendix 1.

According to Collyer & Adams (2018), RRPP is very flexible. It can be used to compare groups as in an ANOVA but also to conduct regression-type analysis of covariates – in other words, it can handle any linear model. It can be applied equivalently to univariate and multivariate data. These considerations are identical to PERMANOVA.

Unlike PERMANOVA, RRPP can access many of the follow-up techniques available to conventional linear models. For example, many of the things you can do with the output of `lm()` can also be done with RRPP output. Collyer & Adams (2018) highlight that it can be used:

- To estimate coefficients from ordinary least-squares (OLS) or generalized least-squares (GLS)
- With type I, II, or III sums of squares
- To estimate effect sizes in several ways
- To analyze mixed models (e.g., ANOVA with both fixed and random effects)

Several authors have used RRPP to explore how morphometrics (the shape of objects). Gagnon et al. (2023) use RRPP to conduct a phylogenetic ANOVA of taxa within the *Solanum* genus to understand patterns in the presence of underground organs. Kucuk et al. (2023) examined the gut bacteria within a scarab beetle. They identified taxa via gene sequencing and then used RRPP to conduct PERMANOVAs of alpha and beta diversity (i.e., variation in the number of taxa within samples and variation in taxa identity among samples).

Theory

RRPP is permutation based but is more computationally complex than the other techniques that we've covered – check out Appendix S1 from Collyer & Adams (2018) if you want to see a lot of matrix algebra! Here, I briefly summarize some steps in this technique as I understand them. Even with this summary, however, RRPP will likely remain more of a black box than any of the other techniques we've covered.

You may recall that a model can be partitioned into two components: fitted values and residuals. For example, consider the following simple model:

$$y \sim x$$

This model implies a 1:1 relationship between x and y . If one of the data points was $\{x = 4, y = 5\}$, the fitted value from this model is 4 and the residual is 1.

To test the effect of a factor, RRPP fits two models, one with and one without the factor. These are termed the full and reduced model, respectively. For example, a model of factor x would use:

$$\text{Reduced model: } y \sim 1$$

$$\text{Full model: } y \sim x$$

The reduced model describes how much variation is explained by other terms – this can be a simple intercept as in this example or could include other variables that you want to account for. The full model describes how much variation is explained by adding factor x .

The reduced and full models are each partitioned via matrix algebra into a matrix of fitted values and a matrix of residuals. The residuals from the reduced model are then permuted (this is the origin of the technique's title) and added back to the fitted values from the reduced model. When these values are added together, they produce random 'pseudovalues' that are used to estimate the variation (SS) explained by the factor in the full model.

The F-statistic for each term is calculated from the ANOVA statistics as usual. Statistical significance is of course assessed via permutations, but in RRPP the likelihood of this F-statistic is calculated by converting it to a Z-score based on the distribution of F-statistic values obtained from the permutations. If desired, the effect size can also be based on other metrics such as the SS accounted for by a term – though, again, it is calculated from the distribution of values of that metric obtained from the permutations.

This technique can be applied to multivariate data or to a distance matrix. In the latter instance, RRPP first applies a Principal Coordinate Analysis (PCoA) to the distance matrix. PCoA is an ordination technique that we'll discuss later, but for our purposes here I'll simply note that this technique simplifies the calculations because the principal coordinates are uncorrelated with one another and therefore permutations only need to be applied to the rows instead of to both the rows

and columns of the distance matrix. One complication that can arise from the use of PCoA is that semimetric distance measures can result in negative eigenvalues. An adjustment is automatically applied to correct for these (see 'Application of RRPP to Semimetric Distances' below for an example of when this matters).

RRPP uses permutations in many parts of an analysis, but uses the same set of permutations throughout an analysis. To increase repeatability of model results, the seed of the random number generator is set by default to the number of permutations – for example, if `iter = 99`, the seed is 100 (i.e., iterations plus the actual data permutation). However, the user can also specify the seed or allow it to be truly random.

Implementation in R (**RRPP::lm.rrpp()**)

RRPP is available only through the RRPP package. If you haven't already installed it on your machine, you will have to do so before you load it.

```
library(RRPP)
```

Data Organization

RRPP requires data in a unique format called 'rrpp.data.frame'. Although it is called a data frame, it is technically a list. This format is necessary so that univariate and multivariate data can be stored as distinct objects within the larger object. It can be created using the function `rrpp.data.frame()`.

These objects are created below for each of our examples.

View the structure of these objects (hint: `str()`) to familiarize yourself with how they are organized.

RRPP::lm.rrpp()

The key function is `lm.rrpp()`. Its usage is:

```
lm.rrpp(  
  f1,  
  iter = 999,  
  turbo = FALSE,  
  seed = NULL,  
  int.first = FALSE,  
  RRPP = TRUE,  
  full.resid = TRUE,  
  block = NULL,  
  SS.type = c("I", "II", "III"),  
  data = NULL,  
  Cov = NULL,  
  print.progress = FALSE,  
  Parallel = FALSE,  
  verbose = FALSE,  
  ...  
)
```

The main arguments are:

- `f1` – model formula such as `Y ~ A + B * C`. The left-hand side of this formula can be a univariate or a multivariate object. All terms should be contained within an object of class 'rpp.data.frame' (see above). If the response is multivariate, it should be expressed as a distance matrix.
- `iter` – number of permutations to conduct. Default is 999.
- `turbo` – if `TRUE`, turns off estimation of coefficients in each permutation and thus accelerates processing speed.
- `seed` – optional argument to control set of random permutations that is conducted. Default (`NULL`) uses a pre-defined seed based on the number of permutations. Any number can be specified as a seed, or "random" to use a random seed.
- `int.first` – whether to test interactions of 'first main effects' before testing subsequent main effects. Default is to not do so.
- `RRPP` – whether to permute residuals from null models for significance testing. Default is to do so (`TRUE`).
- `block` – optional argument to identify blocks so that permutations are restricted to occur within them.
- `SS.type` – type of sums of squares to calculate. See discussion in 'Complex Models' chapter for more details. Three options:
 - `I` – sequential SS. The default.
 - `II` – hierarchical SS
 - `III` – marginal SS
- `data` – where the response and explanatory variables specified in the formula (`f1`) are located. Note that this must be an object of class 'rpp.data.frame'.
- `cov` – optional argument to include covariance matrix

Note that these arguments provide some of the same controls as for `adonis2()`, though the argument names and defaults differ.

The results from `lm.rpp()` are saved to an object of class 'lm.rpp'. This object is a list containing various items, including:

- `LM` – objects produced by linear model, including the data, coefficients, etc.
- `ANOVA` – SS type and information used to construct ANOVA tables.
- `PermInfo` – number of permutations (`perms`), type of residual randomization (`perm.method`), etc.
- `Models` – full set of possible model combinations

Objects of class 'lm.rpp' can be used with many 'downstream' functions. Examples include `anova()`, `plot()`, etc. See below for details.

Simple Example

Applying this technique to our simple example:

```
perm.eg.rpp <- rpp.data.frame(  
  resp.dist = dist(perm.eg[, c("Resp1", "Resp2")]),  
  Group = perm.eg$Group  
)
```

```
perm.eg.fit <- lm.rrpp(
  fl = resp.dist ~ Group,
  data = perm.eg.rrpp)

anova(perm.eg.fit)
```

```
Analysis of Variance, using Residual Randomization
Permutation procedure: Randomization of null model residuals
Number of permutations: 1000
Estimation method: Ordinary Least Squares
Sums of Squares and Cross-products: Type I
Effect sizes (Z) based on F distributions
```

	Df	SS	MS	Rsq	F	Z	Pr(>F)
Group	1	154.167	154.167	0.88179	29.839	2.0429	0.039 *
Residuals	4	20.667	5.167	0.11821			
Total	5	174.833					

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Call: lm.rrpp(fl = resp.dist ~ Group, data = perm.eg.rrpp)
```

Other than slight formatting differences and the addition of a Z statistic, this output is identical to what we obtained from `adonis2()`.

Grazing Example

```
Oak.rrpp <- rrpp.data.frame(
  Oak1.dist = vegdist(Oak1),
  grazing = Oak$GrazCurr
)

Oak.fit <- lm.rrpp(Oak1.dist ~ grazing, data = Oak.rrpp)

anova(Oak.fit)
```

```
Analysis of Variance, using Residual Randomization
Permutation procedure: Randomization of null model residuals
Number of permutations: 1000
Estimation method: Ordinary Least Squares
Sums of Squares and Cross-products: Type I
Effect sizes (Z) based on F distributions
```

	Df	SS	MS	Rsq	F	Z	Pr(>F)
grazing	1	1.0401	1.04009	0.04674	2.2063	3.0428	0.003 **
Residuals	45	21.2135	0.47141	0.95326			
Total	46	22.2536					

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Call: lm.rrpp(fl = Oak1.dist ~ grazing, data = Oak.rrpp)
```

This analysis is consistent with the others we've covered, and indicates that species composition varies with current grazing status. However, see the 'Application of RRPP to Semi-metric Distances' section below for some nuance for how this technique handles semi-metric data.

Using RRPP Analysis Results for 'Downstream' Analysis and Graphing

There are many convenience functions for exploring and using the results of a RRPP analysis. Some examples follow. As is common in R, these functions act in a certain way depending on the class of the object they are given. To search for help about a function like this for a particular class of object, add the object class at the end of the function name. For example, above we used `anova()` to explore 'lm.rrpp' objects. To find help for this function:

```
?anova.lm.rrpp
```

(note that just searching `?anova` won't get you here)

RRPP::anova.lm.rrpp()

The usage of this function is:

```
anova(  
  object,  
  ...,  
  effect.type = c("F", "cohenf", "SS", "MS", "Rsqr"),  
  error = NULL,  
  print.progress = TRUE  
)
```

The main arguments are:

- `object` – an object of class 'lm.rrpp' (i.e., produced from `lm.rrpp()`).
- `effect.type` – which value to calculate effect size from. The effect size is always a Z-score, but it is based on the distribution of values for the given metric. I don't have a good sense of how much the effect sizes vary among metrics. The value that is used is reported in the output – both above the ANOVA table and in the description of the P-value. Options:
 - `F` – use distribution of F values from the permutations. The default.
 - `cohenf` – use distribution of values of Cohen's f-squared from the permutations.
 - `SS` – use distribution of SS values from the permutations.
 - `MS` – use distribution of MS values from the permutations.
 - `Rsqr` – use distribution of partial R-square values from the permutations.
- `error` – optional string to define denominator (MS error term) for calculation of each F statistic in model. This is valuable if you have a hierarchical structure to your design, such as a split-plot design.

While this function is called, you can also index it to extract specific elements such as the F-statistic and P-value for each term:

```
anova(Oak.fit)$table[ , c("F", "Pr(>F)")]
```

```
              F Pr(>F)
grazing    2.2063 0.003 **
Residuals
Total
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This function can also be used to compare different models.

RRPP::summary.lm.rrpp()

The usage of this function is:

```
summary(
  object,
  formula = TRUE,
  ...
)
```

This reports a lot of information on the screen. Even more is hidden and available for indexing – use `str()` to explore it.

RRPP::coef.lm.rrpp()

The usage of this function is:

```
coef(
  object,
  test = FALSE,
  confidence = 0.95,
  ...
)
```

If `test = TRUE`, this function tests whether each coefficient differs from zero. The confidence limit is reported to the desired level specified by `confidence`.

In the current version of this package, I get an error message that coefficient testing was turned off during the `lm.rrpp()` function. However, setting the `turbo` argument to `TRUE` or `FALSE` doesn't appear to change this.

RRPP::plot.lm.rrpp()

The usage of this function is:

```
plot(
  x,
  type = c("diagnostics", "regression", "PC"),
  resid.type = c("p", "n"),
  fitted.type = c("o", "t"),
  predictor = NULL,
  reg.type = c("PredLine", "RegScore"),
  ...
)
```

The key arguments are:

- **x** – an object of class 'lm.rrpp'.
- **type** – which type of plot to create. Three options:
 - **diagnostics** – diagnostic plots for multivariate data.
 - **regression** – multivariate dispersion vs. a predictor value (specified by predictor argument).
 - **PC** – projects data onto eigenvectors, accounting for as much variation as possible in as few dimension as possible.
- **predictor** – covariate to plot. Only used if type = "regression".
- **reg.type** – type of line to plot. Only used if type = "regression". Two options:
 - **PredLine** – prediction line
 - **RegScore** – regression score

These graphics are more useful for regression-type models than for ANOVA-type models.

There is also a `convert2ggplot()` function that attempts to convert an RRPP plot into a ggplot object. This can be useful for customizing graphics.

RRPP::predict.lm.rrpp()

The usage of this function is:

```
predict(
  object,
  newdata = NULL,
  block = NULL,
  confidence = 0.95,
  ...
)
```

Given an 'lm.rrpp' object, this function calculates predicted values for the data specified in **newdata**. The **newdata** object should include all levels of the factors included in the model.

For example, to predict the mean value for each level of the grazing treatment:

```
Oak.DF <- data.frame(grazing = c("Yes", "No"))
row.names(Oak.DF) <- Oak.DF$grazing
```

```
Oak.pred <- predict(Oak.fit, newdata = Oak.DF)
```

The resulting object is of class 'predict.lm.rpp'. It can be plotted or used for other purposes. For example, there is a function to plot an object of this class:

```
?plot.predict.lm.rpp
```

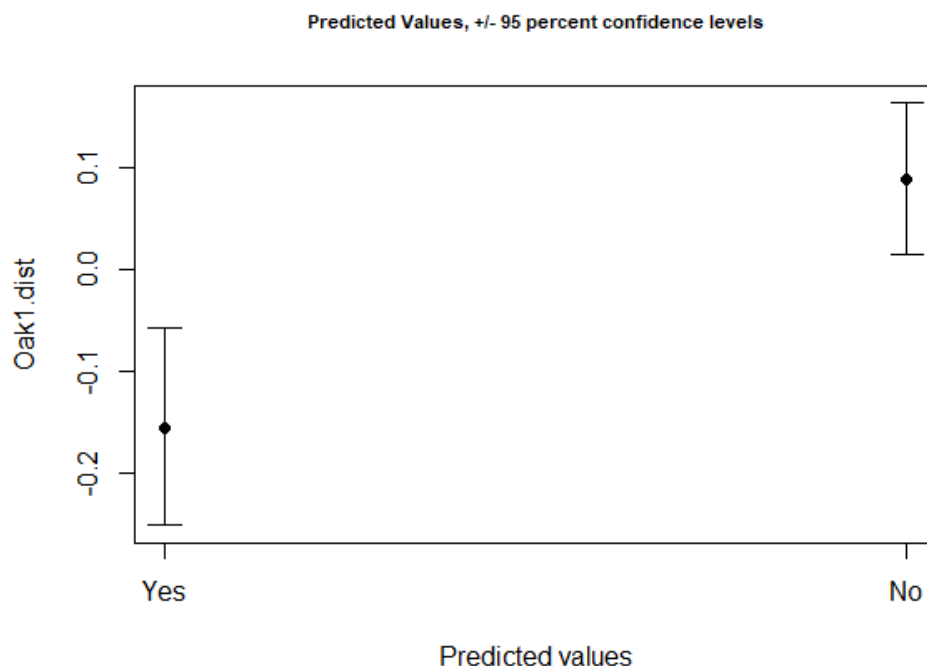
The usage of this function is:

```
plot(  
  x,  
  PC = FALSE,  
  ellipse = FALSE,  
  abscissa = NULL,  
  label = TRUE,  
  ...  
)
```

The key arguments are:

- `x` – an object of class 'predict.lm.rpp'.
- `PC` – whether to rotate the data and display the principal components. Default (`FALSE`) is to show the first two variables. When `TRUE`, the points are shown maximizing the variation explained by these two dimensions.
- `ellipse` – whether to display confidence limits around the mean values. Default is to not do so.
- `abscissa` – vector of values used in predictions, to form horizontal axis of resulting plot.
- `label` – whether to add labels to the points. Default is to do so (`TRUE`).

```
plot(x = Oak.pred, PC = FALSE, ellipse = TRUE, abscissa = Oak.pred$grazing)
```



Predicted values for currently grazed and currently ungrazed plots as determined by RRPP. Values are for the first axis of the PCoA.

Application of RRPP to Semi-metric Distances

In the notes above, we demonstrated how RRPP could be used to analyze how species composition is affected by grazing status:

```
Oak.fit <- lm.rrpp(Oak1.dist ~ grazing, data = Oak.rrpp)
```

```
anova(Oak.fit)
```

```
Analysis of Variance, using Residual Randomization
Permutation procedure: Randomization of null model residuals
Number of permutations: 1000
Estimation method: Ordinary Least Squares
Sums of Squares and Cross-products: Type I
Effect sizes (Z) based on F distributions

      Df      SS      MS      Rsq      F      Z Pr(>F)
grazing    1  1.0401 1.04009 0.04674 2.2063 3.0428  0.003 **
Residuals 45 21.2135 0.47141 0.95326
Total      46 22.2536
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Call: lm.rrpp(f1 = Oak1.dist ~ grazing, data = Oak.rrpp)
```

However, this analysis does not match the results of a PERMANOVA analysis using the default settings of `adonis2()`:

```
grazing.result.adonis2 <- adonis2(Oak1.dist ~ grazing, data = Oak)
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1.dist ~ grazing, data = Oak)
      Df SumOfSqs      R2      F Pr(>F)
grazing    1   0.6491 0.05598 2.6684  0.001 ***
Residual 45  10.9458 0.94402
Total      46 11.5949 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note the difference in the total SS within the two analyses (I've made them bold) – 22.2536 vs. 11.5949. All other differences between these analyses follow from this difference.

Michael Collyer, the lead author of RRPP, indicates that this discrepancy is because the Bray-Curtis distance measure is semi-metric. This means that it does not satisfy the triangle inequality and

cannot be perfectly mapped in a Euclidean space. One way that this is demonstrated is through negative eigenvalues in a Principal Coordinate Analysis (PCoA) – these can be calculated mathematically but are problematic for interpretation.

RRPP automatically adjusts values, forcing semi-metric distances to behave as if they were metric. We can avoid these values in `adonis2()` by including a correction factor which forces the distance matrix to be metric (i.e., to satisfy the triangle inequality). Having done so, the ANOVA table from `adonis2()` is identical to that from `lm.rpp()`.

```
grazing.result.adonis2a <- adonis2(Oak1.dist ~ grazing, data = Oak, add = "cailliez")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1.dist ~ grazing, data = Oak, add = "cailliez")
      Df SumOfSqs      R2      F Pr(>F)
grazing  1   1.0401 0.04674 2.2063  0.002 **
Residual 45  21.2135 0.95326
Total    46  22.2536 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this case, the effect of grazing remains significant but there is no assurance that that will always be the case.

Whether this correction is acceptable will depend on your research foci. I'm not aware of any studies assessing the effect of this correction on statistical conclusions.

Conclusions

RRPP is widely applicable, particularly for datasets where it is appropriate to represent the data using a metric measure such as Euclidean distance. RRPP includes features that are unavailable in other techniques, such as the ability to test random effects and to use many of the follow-up techniques that are available to conventional (univariate) linear models.

Some recent papers that used this technique (e.g., Kucuk et al. 2023) described their approach as doing a PERMANOVA using `RRPP::lm.rpp()`. This language highlights the similarity between RRPP and PERMANOVA.

However, PERMANOVA (`vegan::adonis2()`) provides some additional control when analyzing semi-metric measures. When semi-metric dissimilarity measures are used, RRPP automatically adjusts the dissimilarities so that they behave as if they are metric. Doing so alters the relationships between the data matrix and the dissimilarity matrix: the dissimilarities are not simply based on the data in two sample units but on those data plus some adjustments to make the set of dissimilarities 'behave'. However, it is unclear how much of an effect this has on the conclusions of an analysis.

References

Collyer, M.L., and D.C. Adams. 2018. RRPP: an R package for fitting linear models to high-dimensional data using residual randomization. *Methods in Ecology and Evolution* 9:1772-1779.

Gagnon, E., L. Baldaszi, P. Moonlight, S. Knapp, C.E.R. Lehmann, and T. Särkinen. 2023. Functional and ecological diversification of underground organs in *Solanum*. *Frontiers in Genetics* 14:1231413.

Kucuk, R.A., B.J. Campbell, N.J. Lyon, E.A. Shelby, and M.S. Caterino. 2023. Gut bacteria of adult and larval *Cotinis nitida* Linnaeus (Coleoptera: Scarabaeidae) demonstrate community differences according to respective life stage and gut region. *Frontiers in Microbiology* 14:1185661.

Telemeco, R.S., and E.J. Gangloff. 2020. Analyzing stress as a multivariate phenotype. *Integrative and Comparative Biology* 60:70-78.

Media Attributions

- RRPP.grazing.pred

23. Complex Models

Learning Objectives

- To consider ways to analyze a factor with multiple levels via pairwise contrasts.
- To consider ways to simultaneously analyze multiple explanatory variables.
- To understand the consequences of specifying different types of Sums of Squares.
- To introduce custom functions.

Key Packages

```
require(vegan, RVAideMemoire)
```

Introduction

We have focused thus far on techniques to compare *a priori* groups. However, recall that ANOVA is a specific instance of a **linear model** in which the explanatory variable is categorical. Linear models can be used to compare groups (categorical variables) and to regress one continuously distributed variable on another. The same generalization applies to techniques like PERMANOVA – it can be used to analyze any linear model. This is why PERMANOVA is sometimes referred to more generally as a ‘distance-based linear model’ (DISTLM) or as distance-based redundancy analysis (dbRDA) (Legendre & Anderson 1999; McArdle & Anderson 2001).

Most of the concepts outlined in this chapter also apply to other techniques that can handle multiple factors: the generalized ANOSIM test, Mantel test, and RRPP.

Multi-Factor Models

Since PERMANOVA partitions variation to determine how much is associated with different terms (explanatory variables), one of its key strengths is that it can be applied to complex designs such as those with multiple factors and/or covariates.

Furthermore, PERMANOVA can test whether terms have **additive** effects (think of the main effects in an ANOVA) or **interactive** effects (think of the two-way terms in an ANOVA: the effect of one factor depends on the level of another factor). Interactive effects can occur because of differences in the size and/or the direction of the effect.

When testing complex models with interaction terms, interpretation should be done in a logical manner. Anderson (2015) recommends:

- Fit the full model, including main effects and interaction term(s).
- Examine whether the interaction is significant. If it is significant, test levels of one factor within levels of the interacting factor.
- If the interaction is not significant, examine whether the main effects are significant.

The `adonis2()` function can handle both ANOVA and regression models. Each variable is treated as categorical or continuous based on its class. Therefore, it is helpful to always check that the degrees of freedom in a model output correspond to what you expected. A variable that is categorical will have 1 fewer df than it has levels, whereas a variable that is continuous will only have 1 df.

Model Formulation

As we will see with contrasts below, it is important to identify the correct model formula when dealing with complex designs. This is true not just for `adonis2()` but also for standard statistical functions in R (e.g., `lm()`, `aov()`).

Here are some basic rules for building formulae:

- Formulae are arranged such that the dependent variable or matrix is on the left-hand side (LHS) and the explanatory variable(s) are on the right-hand side (RHS):
Dependent variable or matrix ~ Explanatory variable(s)
- The explanatory variables can be categorical (as in ANOVA) and/or continuous (as in regression). R uses the class of the object to determine whether it is categorical or continuous. The possibility of having both categorical and continuous explanatory variables in the same analysis is why `adonis2()` requires data to be in a data frame rather than a matrix.
- Explanatory variables can be arranged together succinctly using a variety of operators. These permit the creation of complex ANOVA tables without having to type all of the terms out. Some examples are shown in the below table.

Example	Note about operator	Equivalent expanded formula
A	effect of single term	A
A+B	effect of terms independently	A + B
A:B	interaction between terms	A:B
A*B	terms are crossed	A + B + A:B
(A+B+C)^2	'^' indicates the maximum order of the desired interactions.	A + B + C + A:B + B:C + A:C
(A+B+C)^2 - A:B	'-' removes the terms specified after it.	A + B + C + B:C + A:C
A + B %in% A	'%in%' indicates that the term on its left is nested within those on its right.	A + A:B

For more information about building formulae, see `?formula`.

Evaluating Complex Designs

Recall that a PERMANOVA applied to a single variable with Euclidean distances is identical to a conventional ANOVA (or linear model) of that variable. This suggests a three-step process to verify that a design is being analyzed correctly:

1. Analyze a univariate response conventionally
2. Duplicate the conventional analysis using PERMANOVA with Euclidean distances
3. Replace the univariate response with your multivariate response and desired distance measure

We will illustrate this process by repeating our test of the effect of grazing. These principles are also illustrated in the 'Restricting Permutations' chapter.

Step 1: Analyze a Univariate Response Conventionally

Begin with a standard univariate analysis.

This analysis could involve a univariate response whose characteristics you or your statistical advisor are most comfortable with. Perhaps one that satisfies the assumption of normality?

This analysis could involve any analytical function or statistical platform. In R, this might entail fitting a linear model via `lm()` or an ANOVA via `aov()`. However, this conventional analysis could also be conducted in a different software package.

Conduct the analysis and ensure that the resulting ANOVA table is correct:

- Degrees of freedom are appropriate for each term
- All necessary terms and interactions are specified
- F-statistic is being calculated with the correct error term. One way to identify the correct denominator is to consider the expected mean squares (EMS) associated with each term. Each term has an EMS that is a linear combination of one or more of the terms in the model. **The correct denominator for a term is the term whose EMS includes all linear combinations except that of the term itself.** Another way to state this is that the denominator should be the term whose EMS would equal the numerator if the null hypothesis were true (Anderson et al. 2008).

For example, imagine that we were not confident that our approach for analyzing the plant compositional response to grazing was correct. We can start by using `aov()` with a univariate response such as species richness:

```
summary(aov(Oak$SppRich ~ grazing, data = Oak))
```

```
      Df Sum Sq Mean Sq F value Pr(>F)
grazing    1    704    704.0   9.964 0.00285 **
Residuals  45   3180     70.7
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We confirm the details of this analysis:

- Grazing has two levels and is represented by 1 df
- No other terms are required
- The F-statistic is calculated by dividing the MS for grazing by the MS for the residual ($704.0 / 70.7 = 9.96$).

Step 2: Analyze the Univariate Response in PERMANOVA with a Euclidean Distance Matrix

Once we are confident that the experimental design has been correctly incorporated into the conventional analysis, repeat the analysis in PERMANOVA. Use the same univariate response variable and set of explanatory variables, and specify the Euclidean distance measure.

If the PERMANOVA approach is set up correctly, the resulting ANOVA table will be identical to that produced in step 1:

- Same set of terms, with same df and SS for each
- Same F-statistic for relevant terms, other than rounding differences.
- If the PERMANOVA analysis is formulated correctly, the resulting ANOVA table will be exactly the same as that produced above, except for rounding errors. If a different term was used as the error term for a factor, the associated pseudo-*F* statistic would differ from what was calculated in the conventional software.
- P-values will be permutation-based and thus not exactly the same – though in my experience they tend not to vary that much from the parametric values.

We can analyze the effect of grazing on species richness in PERMANOVA, with Euclidean distances.

```
adonis2(Oak$SppRich ~ grazing, data = Oak, method = "euclidean")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak$SppRich ~ grazing, data = Oak, method = "euclidean")
      Df SumOfSqs      R2      F Pr(>F)
grazing  1    704.0 0.18128  9.964  0.002 **
Residual 45   3179.6 0.81872
Total    46   3883.6 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the PERMANOVA table and the earlier ANOVA table have identical values for the df, SS, and F-statistic.

Key Takeaways

If a model is formulated correctly, a PERMANOVA applied to a univariate response variable with Euclidean distances will yield exactly the same statistics as a parametric linear model.

Step 3: Substitute Desired Response and Distance Measure in PERMANOVA

Once we're confident that the analysis is coded correctly in PERMANOVA, simply replace the univariate response variable and Euclidean distance measure with the multivariate set of responses and the distance measure appropriate for your hypothesis. The analysis structure will thus be applied identically to the analysis of this more complex response.

In this case, we analyze the effect of grazing status on species composition as expressed by Bray-Curtis dissimilarities:

```
adonis2(Oak1 ~ grazing, data = Oak, method = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ grazing, data = Oak, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
grazing  1    0.6491 0.05598 2.6684  0.002 **
Residual 45   10.9458 0.94402
Total    46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the result is identical to what we saw in the 'PERMANOVA' chapter. Community composition differs as a function of grazing status.

Testing the permutation-based analysis against a conventional analytical function provides reassurance that the former is functioning as intended.

Choice of Sums of Squares

Analysis is more involved when dealing with a complex design than with a single factor. For complex designs, there are several ways to partition variance. Anderson et al. (2008, p.69-73) provide a nice summary of these types, which are briefly summarized in the following table and in Figure 1.41.

Type	Name	Description	Does Order matter?	Partitioning	by =
I	Sequential	Test terms in order listed	Yes	SS of terms sums to total SS	terms
II	Conditional	Test term after accounting for all other terms except those that contain the term	No	SS of terms may not sum to total SS	
III	Partial	Test term after accounting for all other terms	No	SS of terms may not sum to total SS	margin

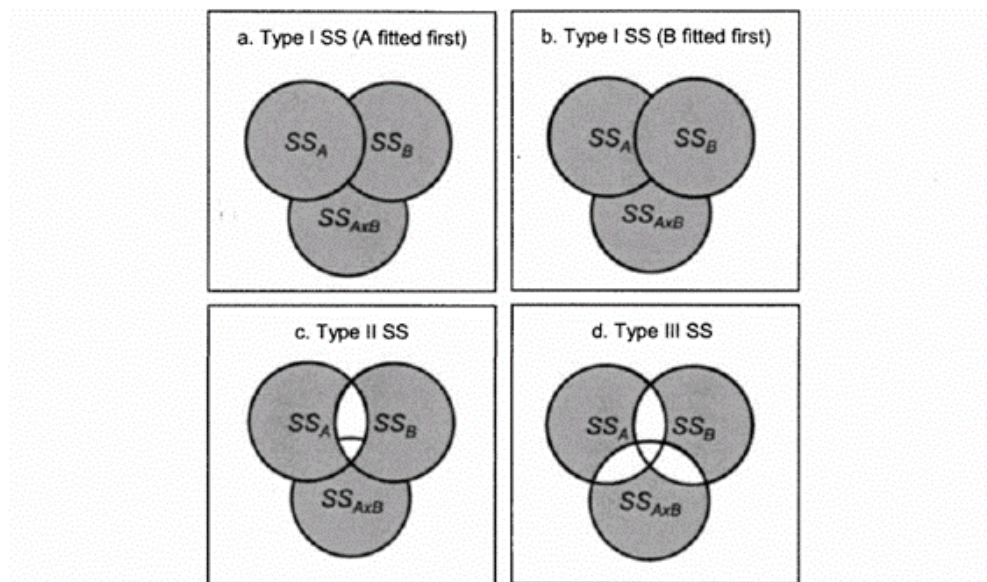


Fig. 1.41. Schematic Venn diagrams demonstrating the conceptual differences in Types of SS for a two-way crossed unbalanced design.

When your data are balanced, it matters little which type of SS you use – the results will be identical in most cases. However, when your data are unbalanced, you have to decide which type of SS is most appropriate based on the questions being explored (Anderson et al. 2008).

Type I or Sequential SS

Type I or Sequential SS are the default in R – both for standard functions like `aov()` and for permutation-based techniques like `adonis2()`. With Sequential SS, each term is fitted after taking into account previous terms in the model. This means that the order of terms in the formula can affect the results. For example, Figure 1.41a shows a model in which term A is fitted before term B, while Figure 1.41b shows the opposite – term B fitted before term A. The same amount of variation is explained, but it's attributed differently. The second term can only explain variation that has not been explained by the first term.

Sequential (Type I) SS may be acceptable or even desirable under some circumstances. For example, it is common to fit covariates first in an ANCOVA, and then to test whether the other term(s) explain significant amounts of the remaining variation. Anderson (2015) notes that the inclusion of a covariate automatically means that the SS from the different terms are not independent, even if your dataset is balanced.

Statisticians generally prefer this type of SS as it fully partitions the variation in a form that is easily understood.

Type II or Conditional SS

Type II or Conditional SS is not described here as this option is not available in `adonis2()`.

Type III or Partial SS

Type III or Partial SS is common in some situations. Each term is fitted after taking into account all other terms in the model, regardless of their position. Thus, the order of the terms does not matter. By accounting for all other terms in the model, Partial (Type III) SS is the most conservative approach – if you detect a significant effect, you would also detect it with any other type of SS, but not *vice versa*.

However, this approach can result in incomplete partitioning of the variance. Consider two explanatory variables that are significantly related to a response when tested individually but that also are closely related to one another. When tested together, it is possible for neither variable to be identified as significant because a sizable amount of the variation could be attributed to either variable and is therefore ignored in this model (this is represented by the white areas within the Venn diagram in Figure 1.41d). Also, the variation that is ignored is also not reported – you can only detect this by checking whether the SS of the terms in the ANOVA table sum to SST.

The current version of this formulation in `vegan` has the surprising action that, if applied to a model with main effects and interaction terms, ignores the main effects and only reports the interaction terms!

Anderson et al. (2008) note that many ecologists and editors expect this approach to be used since it is most conservative. However, in my experience relatively few ecologists realize that their analysis using `aov()`, for example, is not using this type of SS!

Key Takeaways

Deciding which type of sums of squares (SS) to use is important when:

- Testing multiple explanatory variables
- Data are unbalanced

Grazing Example

We have focused thus far on the effect of current grazing, but our dataset also includes information about whether plots were grazed in the past. These two factors (`GrazCurr` and `GrazPast`) are highly unbalanced. Test the effects of these two factors, evaluating how the order of terms and type of SS (I or III) affect the conclusions. Compare the results by evaluating whether the SS explained by the

terms sum to the total SS, and by the broad patterns in statistical significance. How do the results compare?

Model	Type of SS	Results
GrazCurr + GrazPast	I	
GrazPast + GrazCurr	I	
GrazCurr + GrazPast	III	
GrazPast + GrazCurr	III	

For which questions might each model be appropriate?

Model Selection

When choosing among competing complex models, it is helpful to have clear criteria to guide decisions. This is an area of active research, and I think it is likely to only become more prevalent as our computing capabilities and the complexity of our analyses increase (e.g., Mitchell et al. 2017). More complex models often explain more of the variance, but may only incrementally improve model fit.

One way to compare models is to consider how parsimonious they are – whether the improved model fit is ‘worth it’ given the increased complexity. This can be based on Akaike’s Information Criterion (AIC) and variations thereof.

Indices like AIC are not calculated automatically for the models produced from `adonis2()`, but can be calculated by hand. Smaller values indicate more parsimonious models. The actual magnitude of these indices does not matter; what is important is how they compare among different models applied to the same response data. Models with AIC values that differ by < 2 units are generally considered to be equally supported.

Type	Formula	Notes
AIC	$N \log \frac{SSR}{N} + 2p$	Overly liberal (chooses more complex models)?
AICc	$N \log \frac{SSR}{N} + 2p \frac{N}{N - p - 1}$	Corrects AIC for small sample sizes
BIC	$N \log \frac{SSR}{N} + \log(N)p$	Overly conservative (chooses less complex models)?

Formula interpretation:

- SSR = Sum of squares in the residual.
- N = sample size.
- p = number of parameters in model. Number of variables plus 1 for the intercept.

These formulae are from Anderson (2015).

Contrasts (Pairwise and otherwise)

When a factor has only two levels, comparisons among those levels are trivial. The situation becomes more interesting when more than two levels are involved. We will illustrate this using the past and current grazing statuses of each stand. Also, we will focus here on pairwise contrasts; see Appendix 4 for a discussion of other types of contrasts.

A More Detailed Grazing Example

The oak plant community dataset contains factors indicating whether stands were grazed in the past and/or grazed at the time the data were collected (`GrazPast` and `GrazCurr`, respectively). We'll combine these factors together into a new `Grazing` factor (capitalized):

```
Grazing <- factor(with(Oak, paste(GrazPast, GrazCurr, sep = "_")))  
  
summary(Grazing)
```

```
No_No  Yes_No  Yes_Yes  
24      6      17
```

Since each of the original factors is binary, there are four possible levels here. However, there are no stands that were not grazed in the past but were being grazed at the time the data were collected (i.e., the 'No_Yes' combination is absent).

Let's test the overall effect of `Grazing` on vegetation composition, as measured in `Oak1`:

```
Graz.res2 <- adonis2(Oak1 ~ Grazing, method = "bray")
```

View ANOVA table (`Graz.res2`):

```
Permutation test for adonis under reduced model  
Terms added sequentially (first to last)  
Permutation: free  
Number of permutations: 999  
  
adonis2(formula = Oak1 ~ Grazing, method = "bray")  
      Df SumOfSqs      R2      F Pr(>F)  
Grazing  2   0.9095 0.07844 1.8725  0.005 **  
Residual 44  10.6854 0.92156  
Total    46  11.5949 1.00000  
---  
Signif. codes:  
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Yes, there is a relationship between `Grazing` and composition. This means that at least two levels of grazing have different compositions, but doesn't tell us whether some or all of the levels differ from one another. To determine this, we need to use pairwise contrasts.

Pairwise contrasts are planned comparisons between pairs of levels of a factor. If f is the number of levels of a factor, the number of pairs of levels is $f(f-1) / 2$ – the same formula that we used to determine the number of distances in a distance matrix though I've used different symbology here. In this case, there are $3 \times (3-1) / 2 = 3$ pairs of levels:

- No_Yes vs. Yes_No
- No_Yes vs. Yes_Yes
- Yes_No vs. Yes_Yes

The current version of `adonis2()` includes an argument to do one degree of freedom contrasts for a factor:

```
adonis2(Oak1 ~ Grazing, method = "bray", by = "onedf")
```

```
Permutation test for adonis under reduced model
Sequential test for contrasts
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ Grazing, method = "bray", by = "onedf")
      Df SumOfSqs      R2      F Pr(>F)
GrazingYes_No   1    0.1361 0.01174 0.5604  0.958
GrazingYes_Yes   1    0.7734 0.06670 3.1846  0.001 ***
Residual       44   10.6854 0.92156
Total          46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The Grazing factor in the ANOVA table has been replaced by two rows. Each of these rows is a contrast accounting for one df and explaining some of the SS. Together, these rows account for the df and SS that were previously accounted for by Grazing – refer to above results to satisfy yourself that this is the case. The first level in our factor (No_No) was set as the baseline and each of the other levels was compared against it – that's why the names of the other levels are written after the factor name. For example, the row labeled 'GrazingYes_No' is the contrast between No_No and Yes_No, and indicates that these two levels do not differ in composition.

However, this result is not entirely satisfying for several reasons:

- It only shows one set of contrasts – in this case, the contrast between the Yes_No and Yes_Yes treatments is not tested.
- It shows the contrasts in one particular order. Recall that `adonis2()` uses Type I or sequential sums of squares, which means that the first term explains as much of the variation as it can but the second can only explain the variation that was not explained by the first term. In other words, switching the order of the contrasts may change their apparent importance.

We can do better if we control the contrasts ourselves. Although we could code our own contrasts (and do so in Appendix 4), here I modify a function from the `RVAideMemoire` package.

RVAideMemoire::pairwise.perm.manova()

The usage of this function is:

```
pairwise.perm.manova(  
  resp,  
  fact,  
  test = c("Pillai", "Wilks", "Hotelling-Lawley", "Roy", "Spherical"),  
  nperm = 999,  
  progress = TRUE,  
  p.method = "fdr",  
  F = FALSE,  
  R2 = FALSE  
)
```

This function relates a response (*resp*) to a grouping factor (*fact*). It includes numerous options that are not relevant for our purposes – for example, it can do pairwise comparisons for parametric MANOVAs.

The actual code of a function can be seen by calling the function without the parentheses. I display a piece of the code from `pairwise.perm.manova()` here, excerpting it to the sections that apply if the response (*resp*) is a distance matrix:

```
if ("dist" %in% class(resp)) {  
  fun.p <- function(i, j) {  
    fact2 <- droplevels(fact[as.numeric(fact) %in% c(i, j)])  
    resp2 <- as.matrix(resp)  
    rows <- which(fact %in% levels(fact2))  
    resp2 <- as.dist(resp2[rows, rows])  
    vegan::adonis(resp2 ~ fact2, permutations = nperm)$aov.tab[1, Pr(>F)]  
  }  
  ## JDB - additional lines omitted  
  multcomp <- pairwise.table(fun.p, levels(fact), p.adjust.method = p.method)  
  ## JDB - additional lines omitted  
  method <- "permutation MANOVAs on a distance matrix"  
}
```

Each pair of levels needs to be compared in some way. The bulk of the code in this excerpt is the `fun.p()` function, which:

- Subsets both the response and grouping factor to a pair of levels. The levels of the grouping factor are replaced by numbers which can then be paired by indexing them as *i* and *j* (this happens internally so we do not need to define them ourselves).
- Uses `adonis()` to test for differences between the remaining pair of levels
- Returns the p-value of this test

After creating this function, the second-last line applies it within an existing function, `stats::pairwise.table()`, to create a table of pairwise comparisons among the levels of the grouping factor (*fact*).

Customizing the Code: Creating the `pairwise.adonis2()` function

I used the above excerpt of code as a starting point for a new function focused on our needs.

In writing this code, I assumed that:

- `resp` is a distance matrix
- unless otherwise specified, we do not want to make corrections for multiple testing. (If this is desired, the options are described at `?p.adjust.methods`).

I also replaced `adonis()`, which is no longer supported, with `adonis2()`. Here's the function:

```
pairwise.adonis2 <- function(resp, fact, p.method = "none", nperm = 999) {  
  require(vegan)  
  resp <- as.matrix(resp)  
  fact <- factor(fact)  
  fun.p <- function(i, j) {  
    fact2 <- droplevels(fact[as.numeric(fact) %in% c(i, j)])  
    index <- which(fact %in% levels(fact2))  
    resp2 <- as.dist(resp[index, index])  
    result <- adonis2(resp2 ~ fact2, permutations = nperm)  
    result$`Pr(>F)`[1]  
  }  
  multcomp <- pairwise.table(fun.p, levels(fact), p.adjust.method = p.method)  
  return(list(fact = levels(fact), p.value = multcomp, p.adjust.method = p.method))  
}
```

Functions have a standard form. Some notes about this form, using the above function as an example:

- Defined using the `function()` function.
- The object name that `function()` is assigned to is the name of the function that you call to execute it. This function is called `pairwise.adonis2()`.
- The arguments within the parentheses of `function()` are the elements that can be adjusted when the function is executed. If a default value is desired or you wish to choose from a set of options, those can be specified here. For example, I set the default number of permutations to `nperm = 999`.
- The start and end of the function are designated by squiggly brackets (`{ }`) – they may encompass many lines of code. Furthermore, these can be nested within one another, and thus it is typical to indent the code to reflect which set of brackets it is part of. In RStudio scripts, this indenting happens automatically.
- Arguments are referenced within the function's code – everywhere that an argument's name is referenced, it will be replaced with the value that is specified when running the code. In the above code, I've highlighted in red text the places within the function where the arguments are used.

One caveat: this function is hard-coded for a one-way ANOVA ... no other explanatory variables are included in the `adonis2()` function. Can you see how you would adjust it if you needed to include another explanatory variable?

This function is available in the GitHub repository as `pairwise.adonis2.R`. You will need to download and save it into your 'functions' sub-folder, and then load it into R's memory before you can apply it. To apply it to our example:

```
source("functions/pairwise.adonis2.R")

pairwise.adonis2(resp = vegdist(Oak1), fact = Grazing)
```

The output is a list, including the p-values for each pairwise comparison:

	No_No	Yes_No
Yes_No	0.359	
Yes_Yes	0.001	0.932

Our levels are the past and current grazing status, in that order. So, these comparisons indicate that never grazed (No_No) stands differ from always grazed stands (Yes_Yes), but that those that are no longer grazed (Yes_No) do not differ from either of the other combinations.

Other contrasts are also possible. For example, in the 'Custom Contrasts' section of Appendix 4, I demonstrate how these three levels of Grazing can be used to compare stands that were grazed historically (Yes_No and Yes_Yes) with stands that were not (No_No).

Conclusions

Techniques such as PERMANOVA are versatile and can simultaneously test the effects of multiple explanatory models. They can also be extended to answer specific questions, such as contrasts between pairs of levels of a factor.

When data are unbalanced, the type of sums of squares can dramatically affect the conclusions of an analysis. This is equally true for conventional univariate analyses and for permutation-based multivariate analyses.

References

- Anderson, M.J. 2015. *Workshop on multivariate analysis of complex experimental designs using the PERMANOVA+ add-on to PRIMER v7*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK.
- Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK. 214 p.
- Legendre, P., and M.J. Anderson. 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs* 69:1-24.
- McArdle, B.H., and M.J. Anderson. 2001. Fitting multivariate models to community data: a comment on distance-based redundancy analysis. *Ecology* 82:290-297.

Media Attributions

- Anderson.et.al.2008_Figure1.41

24. Controlling Permutations

Learning Objectives

To understand how to replicate analyses by controlling the set of permutations used.

Key Packages

```
require(vegan, permute)
```

Introduction

The purpose of this chapter is to illustrate how to control permutations. This is important primarily so that all aspects of an analysis – including the P -value – are replicable. We've already seen multiple examples of analyses in which the test statistic is identical but the statistical significance varies. Here, we want to consider ways to control which set of permutations is used. Doing so means that the statistical significance will also be unchanged from one run to another, even if those runs are at different times or even on different machines.

These notes are illustrated with analyses using `vegan::adonis2()`. However, the permutation strategies that are developed here can be applied to any of the functions from the `vegan` package that use the same `permutations` argument.

RRPP is coded differently, and has a built-in ability to control permutations. In `RRPP::lm.rpp()`, the set of permutations is identified by the number of permutations requested. As long as you request the same number of permutations, the statistical significance will be the same from run to run. If you change the number of permutations, a different set of permutations is used and the statistical significance may differ.

Grazing Example

We will illustrate the techniques in this section using our oak dataset.

```
source("scripts/load.oak.data.R")
```

The oak plant community dataset contains factors indicating whether stands were grazed in the past and/or grazed at the time the data were collected (`GrazPast` and `GrazCurr`, respectively). As we've done before, we'll combine these factors together into a new `Grazing` factor (capitalized):

```
Grazing <- factor(with(Oak, paste(GrazPast, GrazCurr, sep = "_")))  
summary(Grazing)
```

```
No_No   Yes_No  Yes_Yes  
  24      6     17
```

How to Control Permutations

Permutations are conducted by drawing on a random number generator within R (we won't worry about how it uses this random number generator).

It is sometimes helpful to know exactly which permutations(s) you used so that, for example, you can exactly replicate an analysis. Here are three ways to do so.

`set.seed()`

The `sample()` function was introduced in the chapter about permutations tests. This function simply rearranges the data – it doesn't change the data itself. For example, here is a rearrangement of the numbers from 1 to 10:

```
sample(10)
```

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

If we re-run it, we get a different rearrangement:

```
sample(10)
```

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

To exactly replicate a permutation test, we need to use the same set of permutations and therefore have to start at the same value in the random number generator. The `set.seed()` function allows you to do so. If you set this, run a permutation test, re-set it, and re-run the permutation test, you will get the same permutation(s) each time and thus exactly the same results.

```
set.seed(42); sample(10)
```

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

```
set.seed(42); sample(10) #identical to previous
```

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

```
set.seed(40); sample(10) #different
```

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

```
set.seed(NULL) #reset random number generator  
sample(10)
```

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

Key Takeaways

The `set.seed()` function allows you to specify where to start in the random number generator and thus which set of permutations to use.

Each time you call the random number generator you 'use' a different set of numbers. This means that the `set.seed()` function should be reset before each use. For example, setting it once and then generating two sets results in two different sets:

```
set.seed(42); sample(10); sample(10)
```

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

Resetting it before each set ensures that they're the same:

```
set.seed(42); sample(10); set.seed(42); sample(10)
```

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

In older versions of R (pre-3.6.0), this function gave identical results regardless of platform and in different sessions. Newer versions of R use a different random number generator by default and therefore do not necessarily give identical results unless you change the type of random number generator that is being used. An explanation of this issue is [here](#). You can change the random number generator for an individual action or for an entire session.

Changing one action:

```
set.seed(40, sample.kind = "Rounding"); sample(10)
```

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

This value should match among all of our machines and runs.

Changing the random number generator for the rest of an R session:

```
RNGkind(sample.kind = "Rounding")
```

This function would need to be run once per session, before using `set.seed()`.

Resetting to default random number generator:

```
RNGkind(sample.kind = "default")
```

sample() and permute::shuffle()

You can create a permutation using the `sample()` function, which we've seen earlier, or the `shuffle()` function. The `shuffle()` function has the following usage:

```
shuffle(
  n,
  control = how()
)
```

The main arguments are:

- `n` – how many permuted values to return. Usually set to the number of samples in the object.
- `control` – how the sample units are to be permuted. See chapter about restricting permutations for more information about the `how()` function.

The result is a permutation of the integers from 1:n, just like we saw with `sample()` above. For example, here is a permutation of the sample units in our oak dataset:

```
perm1 <- shuffle(n = length(Grazing))
```

```
[1] 44 36 15 41 35 21 28 26 32 4 47 23 25 17 6 1 46 24 5 8 40 3 42  
[24] 37 29 7 31 39 13 20 12 34 9 14 2 45 38 10 43 22 11 18 30 27 16 19  
[47] 33
```

Note how the integers from 1 to 47 have been permuted. Each integer refers to a position in the original object. For example, the first value (shown in bold) is now in the 16th position.

This permutation can be used to index other objects as occurs in a permutation test. For example, let's use this permutation to index the species richness of the stands. Here's the original species richness data:

```
Oak_explan[, "SppRich"]
```

```
[1] 32 41 35 33 32 51 37 30 29 38 35 33 38 45 41 34 23 22 22 32 29 18 51  
[24] 20 32 36 40 26 34 35 32 18 46 16 38 49 40 33 41 28 38 33 61 44 27 38  
[47] 33
```

It can be confusing to distinguish sets of numbers. These aren't the integers that we saw above – these values don't get below 16 and get as high as 61. The first stand has 32 species (shown in bold).

Here's the permuted species richness data:

```
Oak_explan[perm1, "SppRich"]
```

```
[1] 44 49 41 38 38 29 26 36 18 33 33 51 32 23 51 32 38 20 32 30 28 35 33  
[24] 40 34 37 32 41 38 32 33 16 29 45 41 27 33 38 61 18 35 22 35 40 34 22  
[47] 46
```

By comparing these sets of data, you can verify that the order of the species richness values has been adjusted based on the ordering in `perm1`. For example, the integer 1 is in the 16th position in `perm1` (shown in bold). In the permuted species richness data, the richness from stand 1 is now in the 16th position.

Can you calculate the average richness in each grazing treatment, using both the real data and the ordering in `perm1`?

Start with the real data:

```
Oak_explan |>  
mutate(Grazing = Grazing) |>  
group_by(Grazing) |>  
reframe(SppRich = mean(SppRich))
```



```
# A tibble: 3 × 2
  Grazing SppRich
  <fct>    <dbl>
1 No_No    30.4
2 Yes_No   36.2
3 Yes_Yes  39.6
```

These are the average number of species in each grazing treatment – on average, stands in the Yes_Yes treatment contain the most species and those in the No_No treatment contain the least species.

We could use a set of permutations to assess statistical significance, as we’ve seen throughout this section of the course. Here are the average values based on one permutation:

```
Oak_explan |>
mutate(Grazing = Grazing[perm1]) |>
group_by(Grazing) |>
reframe(SppRich = mean(SppRich))
```

```
# A tibble: 3 × 2
  Grazing SppRich
  <fct>    <dbl>
1 No_No    35.1
2 Yes_No   35.8
3 Yes_Yes   33
```

As expected, the average richness values are all very similar to one another once the data are permuted.

We could have permuted either Grazing or SppRich but we would not have wanted to do both ... do you see why?

permute::shuffleSet()

Since `shuffle()` creates one permutation, you have to call it each time you want to generate a new permutation. The `shuffleSet()` function works just like `shuffle()`, except that it includes an additional argument (`nset`) that allows you to specify how many permutations to create at once.

```
perms <- shuffleSet(n = length(Grazing), nset = 5); perms
```

Note the dimensionality of this object: one row per permutation and one column per sample unit.

This set of permutations could be used in a `for()` loop or a matrix algebra calculation to calculate the results of all permutations. See the ‘Restricting Permutations’ chapter for an example.

For even more control over this set of permutations, it can be proceeded by `set.seed()`. This would ensure that the entire set of permutations is replicable.

25. Restricting Permutations

Learning Objectives

To understand how to replicate analyses by restricting the set of permutations used.

To demonstrate how to conduct permutation tests of complex designs in R.

Readings

Recommended: Anderson & ter Braak (2003).

Key Packages

```
require(vegan, permute)
```

Introduction

The purpose of this chapter is to illustrate how to restrict permutations. This can be necessary for complex designs. Exactly how permutations need to be restricted will depend on the nature of your experimental design. While demonstrating a complex analysis, we will capitalize on the fact that a conventional ANOVA corresponds to a PERMANOVA of a univariate response with Euclidean distances.

In the notes that follow, I refer specifically to `vegan::adonis2()` because you are likely to be using PERMANOVA if you are analyzing a multivariate response in a complex design. However, the permutation strategies that you develop here can be applied to any function from the `vegan` package that use the same `permutations` argument.

The theory of these analyses also applies to `RRPP::lm.rpp()`, though it has some additional built-in capabilities to handle more complex designs. For example, it is possible to analyze a split-plot design in a single model with this function.

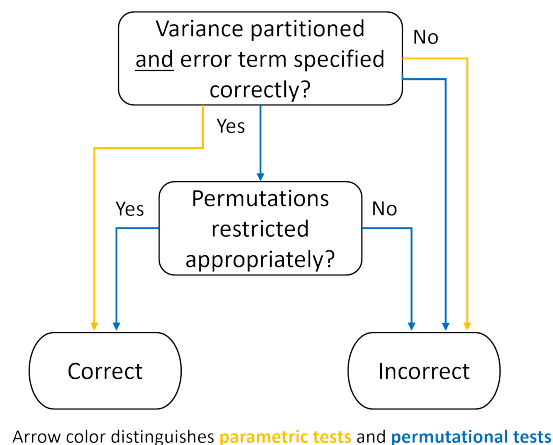
Two Key Issues

In a non-permutational test, the test statistic and its significance are both determined by the terms in the model. For example, if a term is omitted from an ANOVA then the variance and degrees of freedom associated with that term are assigned to the residual. This affects the calculation of the F -statistic for other terms that use the residual as the denominator of the test statistic. And, because the df for a F -statistic includes both the numerator and denominator df, the critical value of the statistic for those other terms will also be affected.

In a permutation test, the test statistic and its significance are somewhat ‘independent’. What I mean by this is that the value of the test statistic and its significance can each be calculated incorrectly. As in a parametric test, failing to include relevant terms in the model can yield an incorrect value for the pseudo- F -statistic. However, the P -value associated with that pseudo- F test statistic is based on the distribution of pseudo- F statistics obtained from permuting the data. Permuting the data incorrectly will give an incorrect P -value, even if the pseudo- F statistic is calculated correctly.

Another way of saying this is that permutation tests include two key decisions:

- **How to calculate test statistic.** Included here are inclusion of the relevant terms in the model and designation of the correct term as the error (denominator) in the calculation of the test statistic. Please note that these considerations are not unique to permutation tests. This is a basic step in all analysis and, like all basic steps, can be ignored or done incorrectly in many different ways.
- **How to restrict permutations.** Permutations are only required for permutation tests, and restricted permutations are only necessary when the design is complex (e.g., multi-factor experiments, hierarchical sampling, etc.). Anderson & ter Braak (2003) provide a readable survey of these ideas, including many worked examples. Somerfield et al. (2021a, 2021b, 2021c) also summarize the permutation schemes for a range of designs – although their articles focus on ANOSIM, the permutations schemes are relevant regardless of the statistical technique being used. I included two helpful tables from these articles in the ‘ANOSIM’ chapter.



Note that there are more ways to incorrectly conduct a permutation test than to incorrectly conduct a parametric test.

These aspects are particularly important when dealing with complex experimental designs, such as where factors are nested. Nesting can take various forms, including:

- **Split-plot designs:** experimental designs in which one treatment is applied to whole plots and

another treatment is applied to split plots (within each whole plot). This approach is very common in agricultural experiments.

- **Repeated measures:** if the same sample units are re-sampled, time can be analyzed as a split plot factor while sample unit is a whole plot factor (Green 1993).
- **Sub-sampling:** if each sample unit is measured in multiple places (e.g., multiple quadrats per plot), the sample units are the error for tests of differences among treatments. Failure to recognize this often results in analyses that treat the quadrats as independent sample units, and is a common source of pseudo-replication (Hurlbert 1984).

We will consider these two decisions, calculating the test statistic and restricting permutations, separately before exploring them in an example.

Calculating the Test Statistic

In any analysis, including the parametric techniques that you've learned about in other courses, it is essential to verify that the variance is being partitioned correctly. This includes i) including all relevant terms in the model, and ii) ensuring that the correct error term is used in the calculation of the test statistic.

Include relevant terms: All necessary terms must be included so that the variance and df are partitioned correctly. In particular, if a term is missing then the variance and df that would otherwise be associated with it are assigned to the residual.

Use the correct error term: Recall that the F statistic from ANOVA (and the pseudo- F statistic from PERMANOVA) is calculated as the ratio of two variances. The numerator of the test statistic for a given term is its mean square, but in complex designs there may be several other terms whose mean squares could form the denominator. However, only one of those terms would be correct to use. For example, in a split-plot experiment, the whole-plot error term forms the denominator for factors applied to the whole plots while the residual or split-plot error term forms the denominator for factors applied to the split plots. If these error terms are not specified correctly, the test statistics may be calculated incorrectly (e.g., whole plot terms that are tested using the residual instead of the whole plot error).

See Anderson & ter Braak (2003), Anderson et al. (2008), and Legendre & Legendre (2012) for details on how to identify the correct exchangeable units for a permutation test. The basic idea is that **the exchangeable units that would form the denominator in a conventional ANOVA are those that should be permuted during a permutation test.**

In some analyses, the appropriate error term is not the residual. The `adonis2()` function currently does not allow you to specify different error terms. One way to resolve this is to save the SS associated with each term in each permutation, and then to manually calculate the pseudo- F statistic for the term of interest afterwards. This is illustrated in the step-by-step analysis of a whole plot factor below.

Note: these ideas can get even more complex when we consider the difference between fixed and random effects.

- A **fixed effect** is one in which we are interested in the particular levels that we included in the study. For example, we might be comparing burned and unburned plots. What we are interested in here is the magnitude of the effect of each level: how different one level is from another.
- A **random effect** is one in which the chosen levels are not of specific interest but rather are a random sample from the population of possible levels. In other words, we want to generalize beyond these particular levels. All models include some estimate of unexplained or error variance (aka residual); random effects add another source of random variation. We may just want to account for this variation, or we may be interested in the size of this variance component relative to other sources of random variation in the model. For example, we might have measured something in multiple quadrats per plot per stand, and be interested in how

much of the variation in this response occurs at each spatial scale. This type of variance component analysis commonly involves hierarchical sampling (Anderson 2015). Unfortunately, the `adonis2()` function is not currently able to analyze random effects.

Restricting Permutations

When we are analyzing a complex design (i.e., one with more than one factor) via permutation tests, it is necessary to ensure that permutations are appropriately restricted. A failure to restrict the permutations appropriately can change the calculated distribution of the test statistic and produce an incorrect *P*-value. Anderson (2015) describes two steps in this process:

1. Determine the exchangeable units
2. Control for other terms that are not being tested and that are not being controlled for due to your choice of permutable units

Exchangeable units

Knowing which term is the denominator of the *F* or pseudo-*F* statistic helps ensure that the variance is being partitioned correctly, but also identifies the exchangeable units that need to be permuted to conduct the correct permutation test. When a term has the residual as its denominator, the exchangeable units are the sample units themselves. If all factors in the model have the residual as the denominator of their test statistic, then the sample units are the exchangeable units and no restricted permutations are required. An example of this would be a fully factorial ANOVA in which the design did not include any blocking or nesting of terms. If another term is the denominator, however, the exchangeable units are not the samples themselves; restricted permutations are required.

For example, in a split-plot design, the unexplained variation among whole plots (i.e., the whole-plot error) is the denominator of the test statistics for factors applied at the whole plot level. This is evident in the fact that the degrees of freedom for the whole plot error term is based on the number of whole plots, regardless of how many measurements were made within them. In a permutation test, variation among whole plots is assessed by restricting permutations such that all observations from the same whole plot are permuted together. Analyses of a split-plot factor – and any interactions with that factor – use the residual as the error term and thus do not require observations to be permuted together. However, they do require restrictions so that permutations occur within rather than across whole plots. They also require inclusion of a term that uniquely identifies each whole plot to account for the variation among whole plots rather than adding this variation to the residual.

Anderson et al. (2008; p. 61-64) describes how to conduct a split-plot analysis in two steps by analyzing the whole plot and split plot components separately. The need to restrict permutations like this may mean that we have to manually synthesize the appropriate terms from each analysis into a single ANOVA table. The analysis of seedling recruitment below provides a worked example.

Controlling for other terms

Complex designs include multiple factors. When assessing the significance of any term, it is necessary to also control or account for the variation that is explained by terms that are not being tested, or that are not controlled for by virtue of the choice of permutable units. This can be done in two ways:

1. Restricting permutations to occur within levels of the other factor(s). Restricting permutations results in an exact test: the Type I error is equal to the *a priori* significance level (e.g., $\alpha = 0.05$).
2. Fitting the other factor(s) first and analyzing (permuting) the residuals. Permuting residuals results in an asymptotically exact test: the Type I error approaches the *a priori* significance level as the sample size increases. This can be done based on the full model, the reduced model, or with the raw data. Currently, the ability to permute residuals is not available in `adonis2()` but is built into `lm.rpp()`.

In practice, this aspect of restricting permutations gets less attention than does the identification of exchangeable units.

Key Takeaways

Analysis of complex models requires careful attention to:

- Inclusion of relevant terms
- Specification of correct error term
- Restrictions of permutations

A Complex (But Realistic) Example: Split-Plot Design

Our oak dataset is not useful for demonstrating restricted permutations as it does not include nesting of different factors. Instead, we will use a dataset about seedling recruitment. This example is modified from Simpson (2014), and is based on data collected by Špačková et al. (1998) and analyzed by Šmilauer & Lepš (2014).

These data are from an experiment testing the effect of four treatments on seedling recruitment. The experiment had a **randomized complete block** design – each treatment was tested within four blocks. The intent was to examine the effects of the treatments; the blocks were a means of ensuring that each treatment was applied under the same range of environmental conditions. In a blocked design like this, the blocks can be very different from one another: they might be in different fields or soil types, for example. This allows the researcher to efficiently test the effects of a treatment under different conditions (blocks). However, each block should be as internally consistent as possible, so that the plots within the block are as similar as possible.

For our purposes, we are going to treat this as a **split-plot** design. This type of design allows different factors to be applied to the blocks (the whole plot scale) and to the plots within the blocks (the split plot scale). Imagine that the four blocks didn't just represent different environmental conditions but received different experimental manipulations: two of the blocks were watered and two were not. Although this is imaginary, doing so allows us to illustrate how this complexity can be incorporated into an analysis.

Each block contained four plots, one plot per treatment:

- Control
- Litter – litter was removed
- Nardus – the dominant species, *Nardus stricta*, was removed
- Li+Mo – litter and moss were removed

In total, there were 16 plots. The experimental design may have looked something like what is shown below, though I suspect the treatments were randomly assigned to the four plots within each block.

Block 1		Block 2	
Control	Litter	Control	Litter
Nardus	Li+Mo	Nardus	Li+Mo

Block 3		Block 4	
Control	Litter	Control	Litter
Nardus	Li+Mo	Nardus	Li+Mo

Schematic of treatments within blocks in the seedling recruitment experiment of Spackova et al. (1998). For illustration purposes, we are also going to assume that two of the blocks were watered and two were not watered.

Seedlings were identified to species and counted in each plot. In total, 23 species were identified.

The data file, 'Spackova.et.al.1998.csv', is available in the GitHub repository. Save this to the 'data' subfolder of your class folder. Open the R project for this course, and load the data:

```
spackova <- read.csv("data/Spackova.et.al.1998.csv", header = T)
```

The object contains 27 columns: a unique ID code for each plot (plot), the treatment it received (treatment), the block it was in (block), and the total number of seedlings counted (seedlsum), followed by the number of seedlings of each of the 23 species.

Create our imaginary water factor, assigning blocks 1 and 2 to the 'Yes' treatment and blocks 3 and 4 to the 'No' treatment:

```
spackova <- spackova %>%
  mutate(water = ifelse(block %in% c("Block1", "Block2"), "Yes", "No"))
```

Verify that this is applied correctly:

```
unique(spackova[, c("block", "water")])
```

```

      block water
1  Block1   Yes
5  Block2   Yes
9  Block3    No
13 Block4    No

```

Before we proceed with this analysis, we need to consider how to restrict permutations.

Restricting Permutations in R: the `permute` package

The `vegan` package uses several functions to define the order in which permutations are conducted. These functions have been compiled in the `permute` package, on which `vegan` depends and which is therefore loaded automatically when `vegan` is loaded. These functions can be called directly if you want to use them in a permutation test of your own creation (see Simpson 2022 for an example) or from within many of the analytical functions in `vegan` via the `permutations` argument. I summarize this approach here; see Simpson (2014, 2022) for details.

The simplest example is one in which all sample units can be freely randomized; in this case, all you need to do is indicate how many permutations you want to conduct. This is done by specifying the number of permutations in the `permutations` argument, as we've done previously. This was also assumed in the `shuffle()` and `shuffleSet()` examples above.

More complex designs, such as repeated measures and split-plot designs, require restricted permutations. Several types of permutation are available (Simpson 2022):

- Free permutation (no restrictions)
- Time series or line transect designs – the order of the observations is preserved; permutations simply start at different positions in the series.
- Spatial grid designs – spatial ordering is preserved in both directions
- Permutation of plots or groups of samples
- Blocking factors – groups of sample units that are not permuted themselves, but that permutations are restricted to occur within. An example of this type of design is provided below.

The `permute` package distinguishes three scales for the purposes of permutations: within, plots, and blocks.

- Blocks are the highest scale. Permutations are restricted so that samples are never permuted among blocks. Blocks must be levels of a factor.
- Plots are the intermediate scale. Samples within a plot can be permuted together, but the plots themselves can also be permuted.
- Within is the lowest scale, and refers to samples nested within plots. If there are no blocks or plots are specified, the permutation instructions specified at this scale are applied to the entire data set.

One limitation of the `permute` package is that it (at least currently) requires that the dataset be balanced. Unbalanced data would have to be manually permuted.

`permute::how()`

The `how()` function is used to specify the details of the permutation design. Its usage is:

```
how(  
  within = Within(),  
  plots = Plots(),  
  blocks = NULL,  
  nperm = 199,
```



```
complete = FALSE,
maxperm = 9999,
minperm = 5040,
all.perms = NULL,
make = TRUE,
observed = FALSE
)
```

The main arguments are:

- **within** – permutation design for samples within levels of plots. Produced using another function, `Within()`, that is described below.
- **plots** – permutation design for plots. Produced using another function, `Plots()`, that is described below.
- **blocks** – factor describing the blocking structure. If blocks are present, sample units are permuted within each block but are NOT permuted across blocks.
- **nperm** – the number of permutations to perform. Default is 199 but analytical functions may specify a different default number: in `adonis2()`, the default is 999.
- **complete** – whether to conduct all possible permutations. Default is to not do so (**FALSE**). Note that you would not want to turn this on if the number of permutations is extremely large.
- **minperm** – minimum number of permutations at which to conduct all possible permutations. Default is 5040.
- **observed** – whether the observed permutation (real data) should be included in the set of all permutations. Default is to not do so (**FALSE**).

`permute::Within()` and `permute::Plots()`

The `Within()` and `Plots()` functions have similar usages:

```
Within(
  type = c("free", "series", "grid", "none"),
  constant = FALSE,
  mirror = FALSE,
  ncol = NULL,
  nrow = NULL
)
```

```
Plots(
  strata = NULL,
  type = c("none", "free", "series", "grid"),
  mirror = FALSE,
  ncol = NULL,
  nrow = NULL
)
```

The key arguments are:

- **strata** – a factor; observations from each level are permuted together. Only applies to `Plots()`.
- **type** – the key argument in both functions; specifies the type of permutation to be conducted. Four options are possible:
 - **free** – samples are freely permuted
 - **series** – used when the ordering of samples needs to be maintained in one dimension. Intended for temporal data or for spatial data such as that collected along a transect.

Adjustments occur by choosing the starting sample, and then maintaining the order of samples that follow. For example, if the numbers (1, 2, 3) are in temporal order, the two possible permutations that maintain this ordering are (2, 3, 1) and (3, 1, 2).

- `grid` – used when the ordering of samples needs to be maintained in two spatial dimensions.
- `none` – no permutations to be conducted.
- `constant` – whether to use the same permutation within each level of `strata`. Only applies to `Within()`.

Creating a Permutation Object

The `how()` and associated functions can be called from within functions such as `adonis2()`, but this can be confusing and would mean that you have to repeat this information each time you ran the analysis. A more efficient approach is to create a new object containing the permutation instructions and then to call that object as needed. For example:

```
CTRL <- how(within = Within(type = "free"),
  plots = Plots(type = "none"),
  nperm = 9999,
  observed = TRUE)
```

In this object, I:

- Specified free permutations of sample units in `within`,
- Increased the number of permutations from the default (199) to 9999, and
- Included the observed data as one of the permutations when calculating the *P*-value.

By including all of the key arguments, even if they haven't been changed, this format is transparent – it is easy to see where changes should be made as appropriate for a given analysis. The `blocks` argument is absent here but is null unless we specify the factor that we want to use to restrict permutations.

One strong advantage of this approach is that it is easy to apply the same permutation regime to multiple analyses. To use this object, I simply specify `'permutations = CTRL'` in the arguments for the desired function. For example, we could call the same permutation object when doing a PERMANOVA and a PERMDISP of the same dataset.

Another advantage of this approach is that it is easy to change elements such as the number of permutations. If we change `nperm` in `CTRL`, the new number of permutations will automatically be applied to all subsequent analyses using `CTRL`. By using this permutation object, we don't have to manually change the number of permutations in each analysis.

Using a Permutation Object

The `check()` function calculates how many permutations are possible for a given dataset and permutation object. With free permutations, there are a large number of ways to permute the 16 observations in `spackova`:

```
check(spackova, control = CTRL)
```

```
2.092279e+13
```

However, this ignores the experimental design. For example, the whole plot scale of this design includes four blocks and the (imaginary) water factor. What are the experimental units? Although it is tempting to think about each of the 16 plots as a separate experimental unit, if the entire block was watered at once then the blocks are the experimental units with respect to that factor. Therefore, there are only four experimental units: two watered, two not watered.

During permutations, we recognize that the blocks are the experimental units by keeping the plots within each block together. Another way to say this is that we restrict the permutations to blocks, and do not permute the plots within blocks. We will create a control object (CTRL.b) that does this:

```
CTRL.b <- how(within = Within(type = "none"),
  plots = Plots(strata = spackova$block, type = "free"),
  nperm = 999,
  observed = TRUE)
```

Compare the order of the explanatory variables in the original data and in a permutation of the data:

```
spackova[ , c("plot", "block", "treatment", "water")]
```

	plot	block	treatment	water
1	rel1	Block1	Cont	Yes
2	rel2	Block1	Litter	Yes
3	rel3	Block1	Nardus	Yes
4	rel4	Block1	Li+Mo	Yes
5	rel5	Block2	Cont	Yes
6	rel6	Block2	Litter	Yes
7	rel7	Block2	Nardus	Yes
8	rel8	Block2	Li+Mo	Yes
9	rel9	Block3	Cont	No
10	rel10	Block3	Litter	No
11	rel11	Block3	Nardus	No
12	rel12	Block3	Li+Mo	No
13	rel13	Block4	Cont	No
14	rel14	Block4	Litter	No
15	rel15	Block4	Nardus	No
16	rel16	Block4	Li+Mo	No

```
spackova[shuffle(nrow(spackova), control = CTRL.b) ,
  c("plot", "block", "treatment", "water")]
```

	plot	block	treatment	water
1	rel1	Block1	Cont	Yes
2	rel2	Block1	Litter	Yes
3	rel3	Block1	Nardus	Yes
4	rel4	Block1	Li+Mo	Yes
13	rel13	Block4	Cont	No

14	rel14	Block4	Litter	No
15	rel15	Block4	Nardus	No
16	rel16	Block4	Li+Mo	No
9	rel9	Block3	Cont	No
10	rel10	Block3	Litter	No
11	rel11	Block3	Nardus	No
12	rel12	Block3	Li+Mo	No
5	rel5	Block2	Cont	Yes
6	rel6	Block2	Litter	Yes
7	rel7	Block2	Nardus	Yes
8	rel8	Block2	Li+Mo	Yes

Verify that the four plots within each block have been permuted as a unit.

Let's explore the implications of this permutation design:

```
check(spackova, control = CTRL.b)
```

24

There are only 4 blocks, so there are only 24 possible permutations (i.e., 4!). There are always fewer permutations at the larger scale of a design than at finer scales (e.g., blocks vs. plots in this example), and thus less power to detect effects at the larger scale.

Return to Our Complex (but Realistic) Example

The seedling data are from an experiment testing the effect of four treatments on seedling recruitment. These treatments were applied within blocks. Our (imaginary) watering treatment was applied to blocks. We will analyze each scale separately.

We will also conduct the analysis in a series of steps. To ensure that each step uses the same permutations, we will re-set the random number generator (`set.seed()`) immediately before each analysis.

Recall that the results of a conventional ANOVA correspond to the results of a PERMANOVA with Euclidean distances. We will use this strategy for verifying the analysis of a complex model as described earlier. We begin by using a simple univariate response, total number of seedlings (seedsum) to understand the analysis details. Once we have worked through the univariate example, we will apply the correct design to the multivariate response consisting of the germination responses of all 23 species.

Whole Plot Analysis (Water and Block Terms)

The whole plot scale of this design includes the blocks and the (imaginary) water factor. For this

portion of the analysis, there are only four experimental units: two watered, two not watered. The treatment term does not relate to variation among blocks so we exclude it from our whole plot model. It will, of course, account for some of the variation within blocks, which we'll analyze in the split-plot portion of this analysis.

We can use PERMANOVA to test the effects of water and block, applying the `CTRL.b` permutation object to permute observations within a block as a unit.

```
set.seed(42)
```

```
adonis2(spackova$seedlsum ~ water + block,
  data = spackova,
  method = "euclidean",
  permutations = CTRL.b)
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Plots: spackova$block, plot permutation: free
Permutation: none
Number of permutations: 24
```

```
adonis2(formula = spackova$seedlsum ~ water + block, data = spackova, permutations = CTRL.b)
      Df SumOfSqs      R2      F Pr(>F)
water    1    517.6 0.02174 0.2682 0.36
block    2    129.1 0.00542 0.0335 1.00
Residual 12 23159.2 0.97284
Total    15 23805.9 1.00000
```

I have emphasized the water and block terms in bold.

We arbitrarily assigned the water factor to blocks, so it's surprising that the water factor accounts for quite a bit of the variation among blocks: $SS_{\text{water}} = 517.6$ whereas the total variation among blocks is 646.7 (i.e., $517.6 + 129.1$).

What is the correct error term for the water factor? There are four blocks so 3 df associated with them. One of these is assigned to the water factor; the other 2 df are associated with other variation among blocks. This is what we need to compare the water factor against. However, the pseudo-F statistic for water that is reported here is incorrect: by back-calculating it using the different terms as its denominator, you can verify that it is based on the residual (i.e., unexplained variation within blocks) rather than the 'block' term (i.e., unexplained variation among blocks).

The correct pseudo-F statistic for the water factor in this design would be:

$$F = \frac{SS_{\text{water}}/df_{\text{water}}}{SS_{\text{block}}/df_{\text{block}}} = \frac{517.6/1}{129.1/2} = 8.02$$

In the current formulation of `adonis2()`, we have to calculate this pseudo-F statistic ourselves. Furthermore, if we just calculate it with the real data we don't have a way to assess its significance. The 'Step-by-Step Analysis' section below illustrates how to do this calculation for the real data and for every permutation, so that we can also determine whether it is statistically significant.

The block term accounts for a small amount of variation ($SS_{\text{block}} = 129.1$); this is the variation among blocks that is not explained by the water factor. Although a pseudo-F statistic and a P-value are

reported for the block term, we are not interested in a statistical test of this term and therefore should ignore them. Remember that this is the residual of the whole plot scale.

The variation that is not accounted for by water or block is reported in the Residual (23159.2). All of this variation occurs within blocks, and is therefore the focus of analyses at the split-plot scale. Before we move to that, however, we'll work through a step-by-step procedure that allows us to calculate the correct pseudo- F statistic for the water factor.

Step-by-Step Analysis (Whole Plot; Manually Calculating pseudo- F -Statistic)

Breaking an analysis into a series of steps can improve our understanding of it. In addition, doing so provides flexibility for situations such as if you wanted to use a term other than the residual as the denominator when calculating a pseudo- F statistic.

In this section, we conduct the whole plot analysis as a series of steps:

1. Create a set of permutations
2. Apply each permutation to the data, and obtain the SS for each term
3. Calculate the pseudo- F -statistic for each permutation
4. Use the distribution of F -statistics to determine the P -value for the term of interest
5. Summarize the data

1. Create a set of permutations

We begin with the `CTRL.b` object created above. This object specifies that blocks are to be freely permuted but that plots within blocks are not permuted.

Now, use `shuffleSet()` to create a set of permutations. The actual ordering of the data is one of the possible permutations, and is the one we are particularly interested in. For simplicity, we will create an object that contains this actual ordering first, followed by multiple random permutations. And, we will re-set the seed of the random number generator first to ensure that we all use the same set of permutations:

```
set.seed(42)

perms <- rbind(1:nrow(spackova),
  shuffleSet(n = nrow(spackova), control = CTRL.b, nset = 999))
#row 1 is the actual data; all others are permutations
```

The above procedure works well when there are many possible permutations, creating an object in which each row is a permutation (actually, real data + permutations) and each column indexes a sample unit. However, in this case there are only 4 blocks and thus 24 permutations. The `shuffleSet()` function identified this and generated the set of all possible permutations. This complicates our object a little bit because it contains the actual ordering of the data twice (verify that this is the case). By viewing the `perms` object, you can verify that the first two rows are identical – in other words, the complete set of permutations begins with the real data. This isn't the case when we generate a subset of the permutations, which is why we include it in `perms`. For this particular situation, we'll drop the second row:

```
perms <- perms[ -2 , ]
```

The resulting object contains all possible permutations of the four blocks, keeping the observations within each block together.

```
head(perms)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]	[,14]	[,15]	[,16]
[1,]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
[2,]	1	2	3	4	5	6	7	8	13	14	15	16	9	10	11	12
[3,]	1	2	3	4	9	10	11	12	5	6	7	8	13	14	15	16
[4,]	1	2	3	4	9	10	11	12	13	14	15	16	5	6	7	8
[5,]	1	2	3	4	13	14	15	16	5	6	7	8	9	10	11	12
[6,]	1	2	3	4	13	14	15	16	9	10	11	12	5	6	7	8

2. Apply each permutation to the data and obtain the SS for each term

We will use a `for()` function to loop through the permutations. For each permutation, we will conduct a PERMANOVA with water and block as terms, and then extract the SS for all terms.

We begin by creating an object where the resulting SS can be stored. We need to anticipate the structure of the analysis to create this `results` object. In this case, there will be four SS (water, block, residual, total; see the PERMANOVA table above), so we will create a column for each and name them in the order they are reported in the results. We create a row in `results` for each permutation to be tested.

```
results <- matrix(nrow = nrow(perms), ncol = 4)
```

```
colnames(results) <- c("water", "block", "residual", "total")
```

Now we loop through the permutations. For each permutation (row from `perms`), we permute the explanatory factors, conduct a PERMANOVA of the response, and save the SS associated with each term to that row of the `results` object.

```
for (i in 1:nrow(perms)) {
  temp.data <- spackova[perms[i, ], ]
  temp <- adonis2(spackova$seedlsum ~ water + block,
    data = temp.data,
    method = "euclidean",
    permutations = 0)
  results[i, ] <- t(temp$SumOfSqs)
}
```

A few notes on this loop:

- The data object is indexed using `perms[i,]` – this is what shuffles the water and block codes so that they are re-assigned to a new row of data.
- The response variable is indexed fully (`spackova$seedlsum`) rather than just calling the column (`seedlsum`). If we did the latter, the response would be found within the `data` argument and therefore would have been permuted just like the water and block terms.
- The number of permutations within each loop is set to zero as we are manually looping

- through the permutations and therefore don't need to do any permutations within each loop.
- The SS from permutation `i` are saved to row `i` of `results`.
- This code uses `adonis2()`. Other functions could be used instead, but the resulting object would have a different structure and therefore a different type of indexing would be required to extract the SS.
- If you want to re-run this loop, you should re-create `results` object first. Otherwise, each new iteration of the loop will add its output to the end of the object that already contains data from a previous run of the loop.

Customizing this code for a different analysis would require adjusting the number and names of columns within `results` and the model formulation within `adonis2()`.

3. Calculate the pseudo-*F*-statistic for each permutation

View the first few rows from `results`:

```
head(results)
```

```
      water  block residual  total
[1,] 517.5625 129.125 23159.25 23805.94
[2,] 517.5625 129.125 23159.25 23805.94
[3,]  52.5625 594.125 23159.25 23805.94
[4,]  76.5625 570.125 23159.25 23805.94
[5,]  52.5625 594.125 23159.25 23805.94
[6,]  76.5625 570.125 23159.25 23805.94
```

The residual SS and total SS are unchanged from one permutation to another. What we have done here has only altered the SS for water and for block.

The test statistic in PERMANOVA is a pseudo-*F*-statistic, which is calculated just like a classical *F*-statistic. The error term for water is the block term – the unexplained variation among blocks – so we calculate the pseudo-*F*-statistic for it as the mean square (MS) for water divided by the mean square for block. We can calculate these for every permutation *en masse*:

```
results <- results |>
  data.frame() |>
  mutate(F.water = (water/1)/(block/2))
```

This calculation of the test statistic using a term other than the residual as the denominator is something that cannot be controlled in the current formulation of `adonis2()`. We could have done this calculation within the loop and saved just the values of this test statistic rather than the SS of each term – I did not do so as I want the analysis to be transparent. It is also more efficient to do this calculation once at the end rather than repeat it for each loop.

I hard-coded the df in the calculation of the pseudo-*F*-statistic. This would have to be customized for other datasets.

The pseudo-*F*-statistic from the real data is in row 1 because it was the first row in our perms object:

```
results[1,] # all data from real data
```



```
      water    block residual    total  F.water
1 517.5625 129.125 23159.25 23805.94 8.016457
```

Verify that these SS match those that we calculated above and that this pseudo- F -statistic agrees with what we expected when we calculated it manually.

4. Use the distribution of F -statistics to determine the P -value for the term of interest

The P -value for the water term is the proportion of permutations with a pseudo- F -statistic as large or larger than that calculated from the real data:

```
with(results, sum(F.water >= F.water[1]) / length(F.water))
```

```
0.125
```

In this case, there are few permutations (24) and even fewer unique combinations (6), and therefore it is mathematically impossible for the p -value to be statistically significant. We could graph the distribution of F -statistics as we've done in other notes, but in this case the low number of permutations makes it pretty boring (give it a try!).

5. Summarize the results

In this analysis, $F = 8.02$ and $P = 0.125$. We conclude that our imaginary water factor has no effect on total seedling recruitment.

Split Plot Analysis (Treatment Term)

Analyses at the split-plot scale focus on terms that vary within the whole plots. However, this doesn't mean that we can ignore the whole plots – if we omit them from our model, the variation they explain will be added to the residual. Furthermore, this is an example of how the design of an experiment should dictate the structure of the analysis.

Although we need to account for the variation among blocks, in this analysis we are not interested in statistical tests of this variation. Therefore we can let the block term represent all variation among blocks. In other words, we don't need to partition the variation due to our imaginary water factor from that which cannot be explained by that factor.

We generally include a blocking term as the first term in a split plot model. This technically doesn't matter if our data are balanced, but is helpful in case the data are unbalanced as it ensures that the blocking term accounts for as much variation as possible when using Type I SS (review the notes on this topic if this seems unfamiliar).

In addition to including the blocking term so that the variation is partitioned correctly, we also need to account for it when permuting the treatments. It may be helpful to view the schematic of the experimental design to demonstrate why this restriction is necessary. If we allowed permutations to occur freely among all plots, permutations could ‘mix’ the variation among treatments within the same block with the variation among plots from different blocks – and the latter variation is not due to treatment alone. Since our intent is to focus on the variation among treatments, we need to restrict the permutations so that plots are permuted within each block, but plots are not permuted across blocks.

Here are two ways to permute the treatment factor without permuting the block factor:

```
CTRL.t <- how(within = Within(type = "free"),
  plots = Plots(type = "none"),
  blocks = spackova$block,
  nperm = 999,
  observed = TRUE)

CTRL.t <- how(within = Within(type = "free"),
  plots = Plots(strata = spackova$block, type = "none"),
  nperm = 999,
  observed = TRUE)
```

The terminology here is a bit confusing, as we are assigning blocks to the `plots` argument. Refer back to the notes about restricting permutations above for an explanation of the arguments within the `how()` function.

These objects are functionally equivalent – they specify that plots are to be freely permuted within blocks but that blocks are not allowed to permute.

There are a much larger number of permutations for this permutation object than for `CTRL.b` above:

```
check(spackova, control = CTRL.t)
```

```
331776
```

Earlier we viewed the original explanatory information and how it was altered by the `CTRL.b` object. We can do the same with this permutation object:

```
spackova[shuffle(nrow(spackova), control = CTRL.t) ,
  c("plot", "block", "treatment", "water")]
```

```
  plot  block treatment water
1  rel1 Block1      Cont   Yes
4  rel4 Block1    Li+Mo   Yes
3  rel3 Block1   Nardus   Yes
2  rel2 Block1   Litter   Yes
6  rel6 Block2   Litter   Yes
8  rel8 Block2    Li+Mo   Yes
7  rel7 Block2   Nardus   Yes
5  rel5 Block2      Cont   Yes
12 rel12 Block3    Li+Mo   No
11 rel11 Block3   Nardus   No
```

```

10 rel10 Block3      Litter      No
9   rel9  Block3      Cont       No
16 rel16 Block4      Li+Mo       No
13 rel13 Block4      Cont       No
15 rel15 Block4      Nardus      No
14 rel14 Block4      Litter      No

```

Verify that treatments are being permuted within blocks but that the blocks themselves have not changed.

Now, we apply the `CTRL.t` object within our analysis:

```
set.seed(42)
```

```

adonis2(spackova$seedlsum ~ block + treatment,
  data = spackova,
  method = "euclidean",
  permutations = CTRL.t)

```

```

Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Plots: spackova$block, plot permutation: none
Permutation: free
Number of permutations: 999

```

```

adonis2(formula = spackova$seedlsum ~ block + treatment, data = spackova, permutations = C

```

	Df	SumOfSqs	R2	F	Pr(>F)
block	3	646.7	0.02716	0.2017	0.051 .
treatment	3	13539.7	0.56875	4.2225	0.051 .
Residual	9	9619.6	0.40408		
Total	15	23805.9	1.00000		

```

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Block is added first to ensure that it accounts for as much of the variation as possible. There are 3 df for block, and this accounts for all of the variation partitioned to the water and block terms in the whole plot analysis. Since the blocks were not permuted, the P -value for this term should be ignored (see above section about testing this term). Gavin Simpson, the author and maintainer of the `permut` package, is uncertain why the P -value from the treatment term is also reported for the block term.

The variation that was unexplained in the whole plot analysis (Residual SS of 23159.3 in that analysis) has now been partitioned into that which can be explained by treatment ($SS_{\text{treatment}} = 13539.7$) and that which cannot ($SS_{\text{Residual}} = 9619.6$). The treatment term is highlighted in bold. It accounts for more than half of the variation in seedling recruitment (partial $R^2 = 0.57$).

The pseudo-F-statistic for the treatment term has been correctly calculated using the residual in its denominator. This term is marginally significant. We can get a more precise estimate of the significance of treatment by increasing the number of permutations – let's do 100,000 permutations now:

```
CTRL.t <- how(within = Within(type = "free"),
```

```
plots = Plots(type = "none"),
blocks = spackova$block,
nperm = 99999,
observed = TRUE)
```

```
set.seed(42)
```

```
adonis2(spackova$seedlsum ~ block + treatment,
data = spackova,
method = "euclidean",
permutations = CTRL.t)
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Blocks: spackova$block
Permutation: free
Number of permutations: 99999
```

```
adonis2(formula = spackova$seedlsum ~ block + treatment, data = spackova, permutations = 0)
      Df SumOfSqs      R2      F Pr(>F)
block    3    646.7 0.02716 0.2017 0.04634 *
treatment 3   13539.7 0.56875 4.2225 0.04634 *
Residual  9    9619.6 0.40408
Total    15   23805.9 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It appears that treatment has a weakly significant effect on overall seedling recruitment.

Summary: Whole and Split Plot Analyses Together

Once we have analyzed each term correctly, we can build a summary ANOVA table which shows how the variance was partitioned for each term. Here, I combine the details of the correct analyses from above.

	df	SS	pseudo- <i>F</i>	<i>P</i>
Whole Plot				
water	1	517.6	8.02	0.125
block	2	129.1		
Split Plot				
treatment	3	13539.7	4.22	0.046
Residual	9	9619.6		
Total	15	23805.9		

In the whole plot analysis portion of this table, the pseudo-*F*-statistic and *P*-value for the water factor are those that we calculated in the step-by-step procedure above. A pseudo-*F*-statistic and *P*-value are not reported for block because we are not testing that term statistically.

In the split-plot analysis portion of this table, we focus just to treatment and the unexplained variation among plots; the block term has already been reported in the whole plot analysis table. We conclude that total seedling germination differs marginally among treatments. If interested, we could use pairwise comparisons to determine which treatments it differs amongst.

Comparison with ANOVA

For comparison, here is a univariate ANOVA of the same data. The `Error()` within the formula is how the `aov()` function specifies that a term other than the residual should be used as an error term.

```
summary(aov(spackova$seedlsum ~ water + Error(block) + treatment,
  data = spackova))
```

```
Error: block
      Df Sum Sq Mean Sq F value Pr(>F)
water   1  517.6    517.6    8.016  0.105
Residuals 2  129.1     64.6

Error: Within
      Df Sum Sq Mean Sq F value Pr(>F)
treatment 3 13540    4513    4.223 0.0403 *
Residuals 9   9620    1069

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can verify that this ANOVA table is identical to what we calculated via PERMANOVA above, except for the *P*-values. There are also a few rounding and formatting differences – for example, I included a row for the total, while the ANOVA table contains the mean square values – but these differences do not alter the interpretation.

Incorrect Analyses

There are many ways to analyze data incorrectly. For example, we could:

- Analyze the entire dataset as a completely random design (no restrictions on permutations, use of residual in error term of all test statistics)
- Ignore the block term while analyzing the effect of treatment
- Restrict permutations incorrectly

The omission of necessary factors or inclusion of unnecessary factors can alter how the variation is partitioned. As discussed above, when a term is omitted, the variation that is associated with it, and the associated df, are assigned to the residual. How much this alters the conclusions will depend greatly on how much variance the omitted term explains. For example, if a term would have explained a lot of variation in the data, then its omission would greatly increase the residual SS and thus would reduce the F-statistic for other terms that used the residual as the denominator in its calculation. If a term was associated with lots of df (e.g., a factor with multiple levels), then its omission will increase the df of the residual and therefore can reduce the residual MS and increase the F-statistic for other terms that use the residual as the denominator.

Restricting the permutations incorrectly can alter the statistical significance associated with a given

value of a test statistic. This could happen if we don't have the permutation object specified correctly or if we call the wrong one (e.g., apply `CTRL.b` in the split plot analysis, or apply `CTRL.t` in the whole plot analysis).

Bottom line: think through the analysis process carefully, and to document your decisions in a script.

Analysis of A Multivariate Response

Switching from a univariate PERMANOVA to a multivariate PERMANOVA is trivial – we simply change the response specified in the model to the desired multivariate response, with the desired distance measure. Here, we'll focus on composition, expressing the number of seedlings of each species as a proportion of the plot total.

Create Response Object

To begin, we extract the data and standardize by plot totals.

```
spackova.comp <- spackova |>
  select(-c(plot, treatment, block, seedlsum, water)) |>
  decostand(method = "total")
```

This is our new response object.

Conduct Whole Plot Analysis

We use the same approach as above, restricting permutations so that blocks are permuted while plots within blocks are permuted together. Since we need to manually calculate the F-statistic for the water term, we'll use the step-by-step procedure shown above.

Note: I had to update the name of the response object within the `for()` loop below – this is shown in **red font**. This is the only element of the code that needed to be changed from what we used in the univariate analysis.

```
set.seed(42)

perms <- rbind(1:nrow(spackova),
  shuffleSet(n = nrow(spackova), control = CTRL.b, nset = 999))
#row 1 is the actual data; all others are permutations

perms <- perms[-2, ] #remove duplicated actual data order

results <- matrix(nrow = nrow(perms), ncol = 4)

colnames(results) <- c("water", "block", "residual", "total")

for (i in 1:nrow(perms)) {
  temp.data <- spackova[perms[i, ], ]
```

```
temp <- adonis2(spackova.comp ~ water + block,
  data = temp.data,
  method = "euclidean",
  permutations = 0)
results[i, ] <- t(temp$SumOfSqs)
}

results <- results %>%
  data.frame() %>%
  mutate(F.water = (water/1)/(block/2))
```

Now we can extract the pseudo-*F*-statistic from the actual data:

```
results[1,] # all data from real data
```

```
      water      block  residual    total  F.water
1 0.1071991 0.1626741 0.9706154 1.240489 1.317961
```

Calculate the *P*-value:

```
with(results, sum(F.water >= F.water[1]) / length(F.water))
```

```
0.6666667
```

Conduct Split-plot Analysis

```
set.seed(42)
```

```
adonis2(spackova.comp ~ block + treatment,
  data = spackova,
  method = "euclidean",
  permutations = CTRL.t)
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Blocks: spackova$block
Permutation: free
Number of permutations: 99999
```

```
adonis2(formula = spackova.comp ~ block + treatment, data = spackova, permutations = CTRL.t)
      Df SumOfSqs    R2    F Pr(>F)
block   3  0.26987 0.21755 1.0891 0.5549
treatment 3  0.22724 0.18318 0.9170 0.5549
```

Residual	9	0.74338	0.59926
Total	15	1.24049	1.00000

Although we included both terms in this model, we focus our attention only on the treatment term.

Summarize Results

Finally, we summarize these results in an ANOVA table:

	df	SS	pseudo- <i>F</i>	<i>P</i>
Whole Plot				
water	1	0.107	1.32	0.667
block	2	0.163		
Split Plot				
treatment	3	0.227	0.92	0.555
Residual	9	0.743		
Total	15	1.240		

In this case, there is no evidence of a compositional difference due to either our imaginary water factor (it would be strange if it was significant!) or among treatments. Given our earlier analysis of total seedling numbers, this implies that the treatments affected overall recruitment but do not favor particular species.

It is ecologically possible that the treatments favor particular species at particular blocks. Statistically, this would be tested by examining a block x treatment interaction. However, this interaction can only be tested if there are replicate plots within blocks. Without that replication, as in this example, the block x treatment interaction functions as the whole plot error term (residual).

Conclusions

The variance is partitioned by including the correct terms in the model and ensuring that the correct term is used as the error term (denominator) when calculating the test statistic.

The *P*-value is determined by restricting permutations appropriately.

How much of a difference these decisions make on your conclusions will depend on the characteristics of the data being analyzed and the terms across which the restrictions occur.

One caveat to keep in mind: the `permute()` function is designed for balanced data. If your data are unbalanced (e.g., different numbers of subsamples in each sample unit), you would have to develop the permutations more carefully. This could be done in several ways, including:

- Permute sample unit IDs and then manually assign those permutation sequences to all subsamples within each sample unit.
- Drop data to create a balanced design. Depending on the dataset, data to drop could be

- selected randomly, by excluding one level of a treatment if that level is missing observations, etc.
- Conduct a Principal Coordinates Analysis (PCoA), retaining all axes, and then calculate the centroid for each sample unit. An advantage of this approach is that it can be applied to any distance measure. For scale-dependent variables such as composition, it would not be appropriate to compare averages if those averages were based on different numbers of subsamples. We'll illustrate this idea when we talk about PCoA.

References

- Anderson, M.J., and C.J.F. ter Braak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.
- Green, R.H. 1993. Application of repeated measures designs in environmental impact and monitoring studies. *Australian Journal of Ecology* 18:81-98.
- Hurlbert, S.H. 1984. Pseudoreplication and the design of ecological field experiments. *Ecological Monographs* 54:187-211.
- Mitchell et al. 2017. Power? Wainwright?
- Simpson, G.L. 2014. Analysing a randomized complete block design with vegan. <http://www.fromthebottomoftheheap.net/2014/11/03/randomized-complete-block-designs-and-vegan/>
- Simpson, G.L. 2022. Restricted permutations; using the permute package. <https://cran.r-project.org/web/packages/permute/vignettes/permutations.html>
- Šmilauer, P., and J. Lepš. 2014. *Multivariate Analysis of Ecological Data Using CANOCO 5*. 2 edition. Cambridge University Press.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021a. A generalized analysis of similarities (ANOSIM) statistic for designs with ordered factors. *Austral Ecology* 46:901-910.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021b. Analysis of similarities (ANOSIM) for 2-way layouts using a generalized ANOSIM statistic, with comparative notes on Permutational Multivariate Analysis of Variance (PERMANOVA). *Austral Ecology* 46:911-926.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021c. Analysis of similarities (ANOSIM) for 3-way designs. *Austral Ecology* 46:927-941.
- Špačková, I., I. Kotorová, and J. Lepš. 1998. Sensitivity of seedling recruitment to moss, litter and dominant removal in an oligotrophic wet meadow. *Folia Geobotanica* 33:17-30. <http://doi.org/10.1007/BF02914928>

Media Attributions

- analysis.flowchart
- Spackova

26. Comparison of Techniques

Learning Objectives

To consider the advantages and disadvantages of multivariate statistical techniques for comparing groups.

Readings

Walters & Coen (2006)

Key Packages

```
require(vegan, RRPP)
```

Introduction

In this chapter, I want to summarize the advantages and disadvantages of the techniques that we've discussed in this section of the course. These opinions are admittedly biased, primarily from the perspective of community ecology. See McCune & Grace (2002), Kenkel (2006), Walters & Coen (2006), Ramette (2007), and Anderson & Walsh (2013) for more information.

Please note that there are other techniques that we have not discussed here (e.g., Wang et al. 2012).

Summary of Techniques

All of these techniques are permutation-based. This means that we do not have to assume (multivariate) normality.

The ability to assess significance is a function of sample size, though the number of permutations is not a limiting factor if you have a reasonable sample size. Knowing to control permutations can ensure that statistical tests are repeatable, producing the same P -value each time they are run.

Some of these techniques are only useful for simple designs, while others can handle complex designs. In the latter cases, it is often important to know how to restrict permutations to reflect your study design. Existing functions to do so assume that designs are balanced.

ANOSIM

ANOSIM (`vegan::anosim()`) is a non-parametric technique based on ranks. It is conceptually similar to a non-parametric correlation test and thus is a special form of the more general Mantel test.

Somerfield et al. (2021b) argued that ANOSIM results are more robust than other test statistics: since this technique is based on rank distances, a monotonic transformation of the dissimilarity matrix will not alter the ranks of the dissimilarities themselves. For example, the conclusions would be the same if based on Euclidean distances or on Manhattan distances (recall that Euclidean distances are Manhattan distances squared).

ANOSIM is commonly used for simple designs. Recent efforts have generalized this method to more complex designs with up to three factors, though it cannot distinguish main effects and interactions among factors (Somerfield et al. 2021a, b, c; see here for examples). These generalizations are not currently available in the current formulation of this technique in R so they would have to be manually coded by the user.

Mantel Tests

Mantel tests (`vegan::mantel()`) assess the linear correlation between two distance matrices derived from different variables on the same sample units. One of the distance matrices can be based on a grouping factor to compare among levels of that factor. However, the two distance matrices can also be produced from continuous variables, allowing a regression-type approach.

Mantel tests can also be used to control for one variable or set of variables while testing for the effect of another variable or set of variables. For example, you could correlate your response distance matrix against the spatial distances among sample units, and then test whether the residuals from that analysis relate to a distance matrix calculated from an explanatory variable or set of variables.

Some authors strongly discourage the use of Mantel tests (e.g., Guillot & Rousett 2013) though solutions have been proposed recently (e.g., Lisboa et al. 2014; Crabot et al. 2019; Somers & Jackson 2022) that claim to address the issues with this type of test. The verdict is still out about the utility of these solutions.

MRPP

MRPP (`vegan::mrpp()`) uses a simple test statistic and is only applicable for simple designs.

PERMANOVA

PERMANOVA (`vegan::adonis2()`) is conceptually very similar to ANOVA and linear regression. It can be applied to both metric distances (e.g., Euclidean) and semi-metric dissimilarities (e.g., Bray-Curtis). It analyzes the distances themselves, regardless of the characteristics of those distances. Since it is based on the distance matrix, it can be applied identically to univariate and multivariate data.

How much variation a dataset contains is in part a function of which distance measure is used to summarize the variation among sample units. This means that PERMANOVA can lead to different conclusions depending on which distance measure is used to summarize the data matrix. However, it can quantify the importance of the main effects of factors and of interactions between factors (Somerfield et al. 2021b).

PERMANOVA can be applied to a wide range of complex models. In the version of PERMANOVA available in R, some aspects of analyses need to be manually coded. For example, when analyzing the whole plot aspect of a split-plot design, we demonstrated how to apply a set of permutations and to use the correct error term when calculating the pseudo-*F* statistic.

PERMDISP

PERMDISP (`vegan::betadisper()`) is different from the other techniques we've covered – it simply characterizes the distance between each sample unit and the centroid of the group to which it belongs. These distances can then be analyzed to help understand whether differences among groups are due to location of the centroids, dispersion around those centroids, or both. Knowing whether dispersion varies among groups can help us interpret statistical results and their ecological significance.

RRPP

RRPP (`RRPP::lm.rrpp()`) is similar to PERMANOVA but explicitly designed for metric distances. It is computationally complex but permits use of many existing functions for downstream applications. RRPP is a linear model and can be applied to a wide range of complex models.

RRPP automatically includes an adjustment that causes semi-metric distances to 'behave' as if they were metric – see the RRPP chapter for an example of this. It's unclear to me how much this adjustment might alter the conclusions of an analysis. At a minimum, if this adjustment was being made in the analysis it should also be applied to the distance matrix before it is subject to ordination, etc.

Comparisons of Techniques

Walters & Coen (2006) provide a nice 'real world' comparison of the utility of these techniques for assessing compositional differences between treatments. They compared two of the techniques we covered, ANOSIM and PERMANOVA, with a classic multivariate analysis of variance (MANOVA) and with ECOSIM, a 'null model analysis of co-occurrence'. All techniques were used to analyze the same dataset evaluating whether composition differed between constructed and natural reefs. Overall, the authors concluded that PERMANOVA was the most sensitive: other techniques did not

detect differences between constructed and natural reefs. They highlighted several strengths of PERMANOVA, including negligible assumptions, flexibility to handle complex designs, and ability to conduct statistical tests even when sample sizes are small.

Anderson & Walsh (2013) compared the sensitivity of ANOSIM, Mantel tests and PERMANOVA to heterogeneity of dispersion. ANOSIM was particularly sensitive to heterogeneity in dispersion, followed by Mantel tests. PERMANOVA was unaffected by heterogeneity in dispersion if the design was balanced, but affected by it if the design was unbalanced (and, the direction of the change depended on whether greater dispersion occurred in the smaller or larger group). Overall, PERMANOVA was more powerful at detecting changes in community structure. They concluded that the combination of PERMANOVA and PERMDISP was particularly helpful for distinguishing location and dispersion effects in balanced designs, though reliability when applied to unbalanced designs can be poor.

Paliy & Shankar (2016) provide a detailed overview of a wide range of techniques from the perspective of microbial ecology. They include ANOSIM, PERMANOVA and Mantel tests, along with a suite of ordination and other techniques. However, they don't do any simulations or provide assessments of how well the techniques perform.

Somerfield et al. (2021b) argue that ANOSIM and PERMANOVA are complementary techniques: ANOSIM can provide an overall test of a factor's importance (robust to transformations of the distances, for example), whereas PERMANOVA can provide insight into elements such as the main effect of a factor and its interactions with other factors, but is sensitive to the details of which distance measure is used.

I haven't seen any papers that have compared RRPP with other techniques, in part because it is so new. In an appendix to their article describing RRPP, Collyer & Adams (2018) compare it to PERMANOVA as coded in the `adonis()` and `adonis2()` functions from `vegan`. They conclude that there is considerable overlap between them and that they yield identical analytical results. However, there are more downstream functions available for the output of RRPP than of PERMANOVA.

Advantages and Disadvantages of Techniques

This table summarizes the advantages and disadvantages of the techniques that we've covered, along with the parametric ANOVA/MANOVA approach.

Method	Advantages	Disadvantages
ANOVA / MANOVA	<ul style="list-style-type: none"> • Powerful • Widely available • Test statistic (F) is familiar, ranges from 0 to infinity 	<ul style="list-style-type: none"> • (Parametric) assumptions rarely met, especially by community datasets • Cannot analyze datasets with more species than samples
Mantel tests	<ul style="list-style-type: none"> • No distributional assumptions • Use to compare two distance matrices from the same set of sample units • Can be used to analyze effects of categorical or continuously distributed explanatory variables • Test statistic (correlation) bounded between -1 and 1 	<ul style="list-style-type: none"> • Limited ability to partition variation among multiple factors (partial Mantel tests) • Can only detect linear correlations • Sensitive to heterogeneity of dispersion
ANOSIM	<ul style="list-style-type: none"> • No distributional assumptions • Non-parametric – based on ranks of distances • Generalized ANOSIM proposed for two-way and three-way designs • Test statistic (R) bounded between -1 and 1 	<ul style="list-style-type: none"> • <code>anosim()</code> has limited ability to partition variation among multiple factors • Continuously distributed explanatory variables can only be analyzed if converted to ordinal factors • Based on rank distances – doesn't use all information • Lower power (high probability of Type II statistical error) if strong gradients are present in data (Gotelli & Ellison 2004) • Very sensitive to dispersion • Generalized ANOSIM not currently available in R
MRPP	<ul style="list-style-type: none"> • No distributional assumptions • Can compare groups of varying numbers of sample units • Test statistic (δ) ranges from 0 to infinity 	<ul style="list-style-type: none"> • Can only analyze differences among discrete groups • Have to choose a weighting method • Limited ability to partition variation among multiple factors (blocked MRPP)
PERMANOVA	<ul style="list-style-type: none"> • No distributional assumptions • Can be used to analyze effect of categorical or continuously distributed explanatory variables • Powerful – multiple factors, unbalanced designs, etc • Can be applied to any distance measure • Test statistic (pseudo-F) ranges from 0 to infinity • Familiar ANOVA structure to results 	<ul style="list-style-type: none"> • Somewhat limited availability
PERMDISP	<ul style="list-style-type: none"> • No distributional assumptions • Natural follow-up to significant ANOVA-type comparisons (i.e., among groups) to test whether groups differ in dispersion 	<ul style="list-style-type: none"> • Only appropriate for categorical explanatory variables

Method	Advantages	Disadvantages
RRPP	<ul style="list-style-type: none"> • No distributional assumptions, but intended for metric distance measures • Powerful – multiple factors, unbalanced designs, random effects, etc. • Test statistic (Z) ranges from 0 to infinity • Familiar ANOVA structure to results • Multiple functions available for ‘downstream’ usage 	<ul style="list-style-type: none"> • Only available through RRPP package • If applied to semimetric distance measures, automatically adjusts for negative eigenvalues • Very new – pros and cons still being established

Conclusions

In these, as in all of the other multivariate techniques that we will discuss this quarter, it is very important that you use appropriate data adjustments (removal of rare species, transformations, and, especially, relativizations) and distance measure. All of these adjustments need to be clearly noted when describing an analysis.

Furthermore, we often apply several techniques to the same data. For example, we might use PERMANOVA to test differences among groups, PERMDISP to test for differences in dispersion, and PCoA or another ordination technique to visualize those differences. **If you are applying several techniques to the same data, it is essential that the same data adjustments and distance measure be used in all cases.** Otherwise, the different techniques (e.g., analysis and visualization) are not using the same data!

Regardless of which technique we’ve used, a significant effect by itself does not allow us to say how the treatments differ. It does not make sense to describe one group as having ‘larger’ or smaller values than another when they are being compared with respect to multiple responses ... can you see why this is so? This is different from how univariate analyses are often interpreted.

There are several ways to understand a significant group comparison test:

- PERMDISP helps distinguish differences in location from differences in dispersion
- Techniques that visualize the data (e.g., ordinations) can give valuable insights into how the treatments differ
- Follow-up tests can explore whether individual response variables differ among the groups. For compositional data, these techniques include SIMPER, Indicator Species Analysis, and TITAN.

References

- Anderson, M.J., and D.C.I. Walsh. 2013. PERMANOVA, ANOSIM, and the Mantel test in the face of heterogeneous dispersions: what null hypothesis are you testing? *Ecological Monographs* 83:557-574.
- Collyer, M.L., and D.C. Adams. 2018. RRPP: an R package for fitting linear models to high-dimensional data using residual randomization. *Methods in Ecology and Evolution* 9:1772-1779.
- Crabot, J., S. Clappe, S. Dray, and T. Datry. 2019. Testing the Mantel statistic with a spatially-constrained permutation procedure. *Methods in Ecology and Evolution* 10:532-540.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.

- Guillot, C., and F. Rousett. 2013. Dismantling the Mantel tests. *Methods in Ecology and Evolution* 4:336-344.
- Lisboa, F.J.G., P.R. Peres-Neto, G.M. Chaer, E.d.C. Jesus, R.J. Mitchell, S.J. Chapman, and R.L.L. Berbara. 2014. Much beyond Mantel: bringing Procrustes Association Metric to the plant and soil ecologist's toolbox. *PLoS ONE* 9(6):e101238.
- Kenkel, N.C. 2006. On selecting an appropriate multivariate analysis. *Canadian Journal of Plant Science* 86:663-676.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Paliy, O., and V. Shankar. 2016. Application of multivariate statistical techniques in microbial ecology. *Molecular Ecology* 25:1032-1057.
- Ramette, A. 2007. Multivariate analyses in microbial ecology. *FEMS Microbiology Ecology* 62:142-160.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021a. A generalized analysis of similarities (ANOSIM) statistic for designs with ordered factors. *Austral Ecology* 46:901-910.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021b. Analysis of similarities (ANOSIM) for 2-way layouts using a generalized ANOSIM statistic, with comparative notes on Permutational Multivariate Analysis of Variance (PERMANOVA). *Austral Ecology* 46:911-926.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021c. Analysis of similarities (ANOSIM) for 3-way designs. *Austral Ecology* 46:927-941.
- Somers, K.M., and D.A. Jackson. 2022. Putting the Mantel test back together again. *Ecology* 103:e3780.
- Walters, K., and L.D. Coen. 2006. A comparison of statistical approaches to analyzing community convergence between natural and constructed oyster reefs. *Journal of Experimental Marine Biology and Ecology* 330:81-95.
- Wang, Y., U. Naumann, S.T. Wright, and D.I. Warton. 2012. mvabund– an R package for model-based analysis of multivariate abundance data. *Methods in Ecology and Evolution* 3(3):471-474. <https://doi.org/10.1111/j.2041-210X.2012.00190.x>

PART III

CLASSIFICATION

Introduction

Thus far, we've covered some essential background material (data adjustments, matrix algebra, distance measures) and how to compare observations from *a priori* groups (PERMANOVA and other techniques). The next important topic that we will cover is how to identify groups *a posteriori* – in other words, how to classify observations into groups.

Again, we will focus on techniques that are distance-based.

We'll focus on three broad categories of techniques:

- **Cluster analysis** – to identify groups (clusters) of observations with similar sets of responses. For more detailed information, see Legendre & Legendre (2012, ch. 8) and Borcard et al. (2018, ch. 4). Cluster analyses can be done hierarchically or for a pre-specified number of groups (*k*-means). The groups that are produced can be used in a variety of ways.
- **Discriminant analysis** – to assess how well observations were classified into groups, and how many were misclassified. This is the focus of chapter 8 of Manly & Navarro Alberto (2017). DA can be conducted using a linear model or based on distances.
- **Classification and regression trees** – to classify observations into groups based on a response (or matrix of responses) together with one or more explanatory variables. We will introduce this concept using a univariate response (De'ath & Fabricius 2000) and then extend it to a multivariate response (De'ath 2002). Random forests are an extension of this technique that are often used for univariate responses with large sample sizes.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- De'ath, G., and K.E. Fabricius. 2000. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology* 81:3178-3192.
- De'ath, G. 2002. Multivariate regression trees: a new technique for modeling species-environment relationships. *Ecology* 83:1105-1117.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.

27. Types of Cluster Analyses

Learning Objectives

To appreciate the range of types of cluster analyses available, based on whether they are hierarchical or not, agglomerative or divisive, polythetic or monothetic, and sequential or simultaneous.

Introduction

Cluster analysis seeks to classify observations into groups such that each observation is more similar to the other observations in its group than to observations in other groups. **This is a descriptive process, not one in which hypotheses are tested.** A cluster analysis can be applied to any dataset, whether or not there is an underlying gradient(s) or structure to the data.

Cluster analysis is based entirely on the response variables; **explanatory variables are not required.** This is in contrast to classification and regression trees, which we will discuss separately.

In the types of cluster analysis considered here, the resulting clusters are assumed to be discrete – each sample unit is assigned to one (and only one) group. However, it's worth remembering that data and communities often vary continuously.

On a related point, “not all problems are clustering problems. Before engaging in clustering, one should be able to justify why one believes that discontinuities exist in the data; or else, explain that one has a practical need to divide a continuous swarm of objects into groups” (Legendre & Legendre 2012, page 338).

Cluster analyses are subject to the same considerations that affect all other analyses, including:

- What sampling strategy will you use (or was used to obtain your data)? Note that representative samples can be acceptable when conducting a cluster analysis; you do not need random samples.
- What measurements will be made? Will you sample all species or a subset of them?
- Will you standardize / transform data? If so, how?
- Which distance measure to use? Choose one that best preserves information from the system being studied.

There are many types of cluster analyses and no single decision tree to easily distinguish them all. Instead, it's more helpful to answer several questions.

Hierarchical or Non-hierarchical?

In a **hierarchical** analysis, groups are composed of subgroups. Thus, the structure at a given level is constrained by the structure at other levels in the dendrogram. A phylogenetic tree is a classic example of this type of structure: *Pseudotsuga menziesii* is within the genus *Pseudotsuga*, which is within the family Pinaceae, etc. While community-level data may not be hierarchical to this degree, this type of approach is quite common. When folk refer to a 'cluster analysis', they often mean a hierarchical approach, as illustrated by a [dendrogram](#).

In a **non-hierarchical** analysis, observations in the same group at one 'level' of the analysis may or may not occur in the same group at another level. The goal is to optimize the structure at a given level irrespective of what the structure might be like at another level. A common example is *k*-means clustering. Since this is non-hierarchical, a dendrogram is not produced.

Does It Build Groups Up or Tear Them Apart?

Hierarchical cluster analyses can move from simplicity to complexity or from complexity to simplicity. This is analogous to how a stepwise regression can be done forward or backward, though each clustering approach is a different type of technique. Like a stepwise regression, these approaches can yield different conclusions.

Agglomerative methods begin with each sample unit assigned to its own cluster and then iteratively fuse (combine) the two most similar clusters, continuing until there is just a single cluster. Distances between clusters are recalculated at each stage. In R, agglomerative clustering can be performed using the `stats::hclust()`, `cluster::agnes()`, and `mclust::mclust()` functions, among others. When folk refer to a 'cluster analysis', this is often what they mean.

Divisive methods begin with a single cluster and then systematically divide the cluster into subgroups. TWINSpan is a famous example of a hierarchical divisive method (Hill et al 1975; McCune & Grace 2002, ch. 12), though it's rarely used at present (Zeleny 2015). In R, divisive clustering can be performed using the `cluster::diana()` and `cluster::mona()` functions. Classification and regression trees are also a divisive technique.

Based On All Available Data, Or On One Variable At A Time?

Polythetic methods are multivariate, and are often based on a similarity or dissimilarity matrix.

Monothetic methods examine all variables to identify the single variable that will best separate groups. These techniques are often divisive rather than agglomerative.

What Is The Order Of Operations?

Sequential – solution is reached through a series of steps. For example, a hierarchical technique often will:

- Identify and group the nearest two sample units
- Compare this group with the other sample units to identify the next most similar pair of objects

- Repeat this process until all sample units were included in a single group.

Similarly, a non-hierarchical technique can start with random centroids for groups and then iteratively adjust them until some stopping criterion is met.

Simultaneous – solution is reached in a single step. Uncommon.

Conclusions

The broad category of 'cluster analysis' encompasses a large range of techniques. The most influential differences are between hierarchical and non-hierarchical techniques and between agglomerative and divisive techniques.

References

- Hill, M.O., R.G.H. Bunce, and M.W. Shaw. 1975. Indicator species analysis, a divisive polythetic method of classification, and its application to a survey of native pinewoods in Scotland. *Journal of Ecology* 63:597-613.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Zelený, D. 2015. TWINSpan in R. <https://davidzeleny.net/blog/2015/05/10/twinspan-in-r/>

28. Hierarchical Cluster Analysis

Learning Objectives

To consider ways of classifying sample units into hierarchically organized groups, including the construction of a dendrogram and the choice of group linkage method.

To identify the number of groups present in a set of sample units.

To begin visualizing patterns among sample units.

Key Packages

```
require(stats, vegan, cluster, dendextend)
```

Introduction

Hierarchical cluster analysis is a distance-based approach that starts with each observation in its own group and then uses some criterion to combine (fuse) them into groups. Each step in the hierarchy involves the fusing of two sample units or previously-fused groups of sample units. This means that the process of combining n sample units requires $n-1$ steps.

We begin by illustrating the process of conducting a cluster analysis.

A Simple Example

Step-by-Step Procedure

As an example, we'll conduct a cluster analysis on five sample units. Our objective is to combine sample units that are most similar together. We begin by converting the data matrix (not shown) into a dissimilarity matrix:

	1	2	3	4
2	26			
3	32	14		
4	42	56	50	
5	56	54	36	10

Recall that this is a symmetric square matrix, but for simplicity we display it as a lower triangle and do not show the first row and last column. This technique can be applied to any type of distance or dissimilarity measure. Since this is a dissimilarity matrix, larger values indicate sample units that are more dissimilar and smaller values indicate sample units that are more similar to one another.

Identify the pair of sample units that are most similar (i.e., have the smallest distance), and fuse (combine) those sample units into a group. In this case, the smallest distance is $d_{5,4} = 10$, and sample units 4 and 5 are combined. The distance at which this fusion occurred (10, in this case) is the **fusion distance**.

Sample units 4 and 5 were each some distance from the other sample units. For example, $d_{4,1} = 42$ and $d_{5,1} = 56$. How far is sample unit 1 from the group containing sample units 4 and 5? There are several ways to decide this (see 'Group Linkage Methods' below). For now, we'll set this distance to the smaller of the two distances: $d_{4/5,1} = 42$. Following the same process, we identify the distance from the combined sample of units 4 and 5 to each of the other sample units in the dissimilarity matrix. Using this information, we update the dissimilarity matrix:

	1	2	3
2	26		
3	32	14	
4/5	42	54	36

Two notes about this dissimilarity matrix:

- Dissimilarities that do not directly involve the combined sample units are unchanged.
- The dimensionality of the dissimilarity matrix has been reduced from 5×5 to 4×4.

Now, we repeat the above steps: identify the smallest distance, fuse those observations together, and recalculate distances from the new group to other sample units. In this case, the smallest distance is $d_{3,2} = 14$, and sample units 2 and 3 are combined. Recalculate all distances as before and update the dissimilarity matrix, reducing the dimensionality to 3×3:

	1	2/3
2/3	26	
4/5	42	36

Repeat one more time, reducing the dimensionality to 2×2:

	1/2/3
4/5	36

The final fusion joins groups 1/2/3 and 4/5, so that all sample units are in a single cluster. In this example, that occurs at a distance of 36.

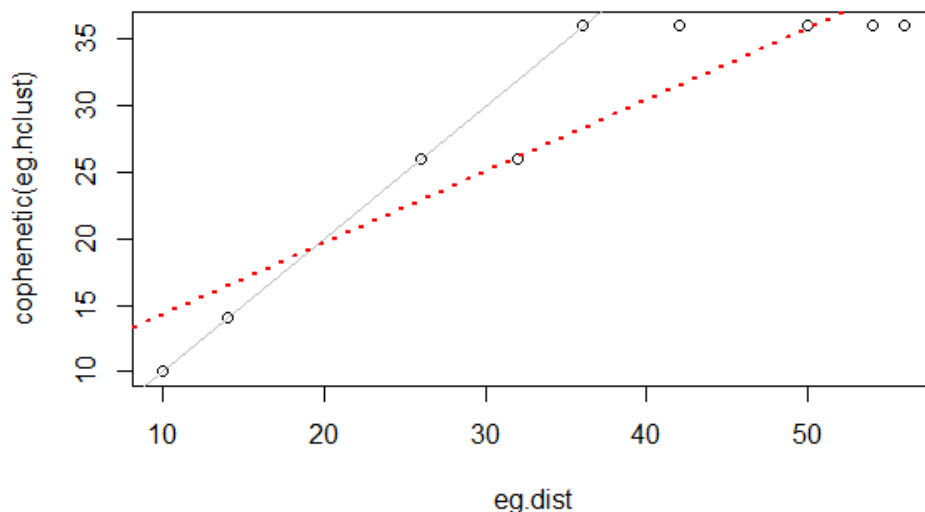
The Fusion Distance Matrix

The fusion distances (aka **cophenetic distances**) can be summarized in a **fusion distance matrix**:

	1	2	3	4
1				
2	26			
3	26	14		
4	36	36	36	
5	36	36	36	10

The fusion distance matrix is a numerical summary of the cluster analysis. It has the same dimensionality as the original distance matrix. Fusion distances are repeated wherever multiple sample units are combined at the same fusion distance.

We can assess how well a clustering method matches the actual dissimilarities by the cophenetic correlation, the correlation between the fusion distance matrix and the original (data-based) distance matrix. In this simple example, the correlation is high (0.92). We can also see this visually by plotting the actual distances among sample units against the fusion distances:

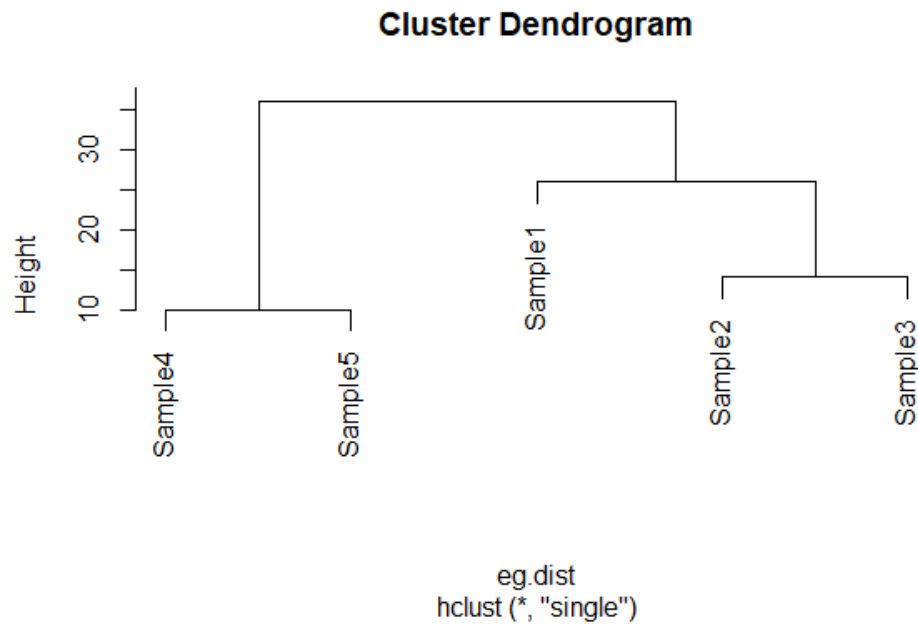


Relationship between actual dissimilarities among sample units (horizontal axis) and the distances at which they were fused in a hierarchical cluster analysis (vertical axis). Note that the fusion distances (aka cophenetic distances) cannot be larger than the actual dissimilarities among sample units; the 1:1 line is shown in grey. The dashed red line is the linear regression between the two sets of points.

Cophenetic correlations from different cluster analyses of the same data can be compared to determine which one fits better. This can be done via a Mantel test or other methods.

The Dendrogram

The fusion distance matrix is often shown visually as a **dendrogram**:



Dendrogram of five sample units from our step-by-step example.

In a dendrogram, the height is the distance at which two sample units or groups of sample units are fused. It is not necessarily the actual distance between them! Refer back to cophenetic distances above if this does not make sense.

McCune & Grace (2002) note that “there is no inherent dimensionality in a dendrogram” (p. 82). The following points are important to keep in mind when interpreting a dendrogram:

- A dendrogram has only one axis. The above example is oriented vertically, but it is also possible – and perfectly appropriate – to orient it horizontally or even circularly.
- The branches may be in no particular order. McCune & Grace (2002, Fig. 10.3) use the helpful **analogy of a mobile to describe a dendrogram**: the branches are the arms of the mobile and thus each branch can be freely reversed regardless of the order of sample units in other branches. However, sample units are generally arranged such that branches do not cross within the dendrogram.
- The lengths of the segments that parallel the axis are proportional to the values along that axis.
- The segments perpendicular to the axis are of arbitrary length, but are generally drawn to improve the clarity of the dendrogram. For example, they are generally drawn so that sample units are equally spaced within the dendrogram.

Chaining occurs when single sample units are added to existing groups (i.e., groups aren’t combined together). Some chaining is ok, but if the entire dendrogram is chained the cluster analysis has found no structure in the dataset. McCune & Grace (2002) suggest avoiding dendrograms with > 25% chaining.

R Script

The above example distance matrix is available in the GitHub repository. Once you've downloaded it to your 'data' sub-folder and opened your R project, you can use the below code to conduct the cluster analysis described above.

```
eg <- read.table("data/cluster.example.txt")

eg.dist <- as.dist(eg) #change to class 'dist'

eg.hclust <- hclust(eg.dist, method = "single") #cluster analysis

cophenetic(eg.hclust) #fusion distance matrix

cor(eg.dist, cophenetic(eg.hclust)) #cophenetic correlation

lm(cophenetic(eg.hclust) ~ eg.dist) #slope and intercept

plot(eg.dist, cophenetic(eg.hclust))
  abline(a = 0, b = 1)
  abline(a = 8.8766, b = 0.5405, col = "red", lty = 3, lwd = 2)

plot(eg.hclust) #dendrogram
```

Group Linkage Methods

Hierarchical polythetic agglomerative clustering requires a method of deciding when to add another sample unit to a group. In our simple example, we combined samples that had the smallest distance. This is called a single linkage method. However, many group linkage methods exist. McCune & Grace (2002, Table 11.1) show that many of these methods are variations of the same combinatorial equation.

Many of the common group linkage methods can be described verbally:

- **Single linkage** (aka nearest neighbor or 'best of friends' or 'friends of friends') – an element is added to a group if it is more similar to any single element in that group than to the other elements in the dataset.
- **Complete linkage** (aka farthest neighbor or 'worst of friends') – an element is added to a group if it is more similar to all other sample units in that group than to other sample units in the dataset. Thus, it becomes more difficult for new sample units to join a group as the group size increases.
- **Group average** – distance between groups is the average of all distances for all pairs of elements
- Median (aka Gower's method)
- Centroid – distance between groups equals the distance between the centroids
- **Ward's minimum variance method** (Ward 1963) – minimizes the error sum of squares (same objective as in ANOVA). The R help files describe this method as aiming 'at finding compact, spherical clusters'. At the beginning, each sample unit is in its own cluster, and therefore the sum of the distances from each sample unit to its cluster is zero. At each subsequent step, the pair of objects (sample units or cluster centroids) is selected that can be fused while increasing the sum of squared distances as little as possible. This method requires Euclidean distances and is technically incompatible with Bray-Curtis dissimilarity (though people use the two together anyways).
- **Flexible beta** – a variant of the combinatorial equation. McCune & Grace (2002) recommend using it with $\beta = -0.25$

- McQuitty's method

I have highlighted in bold the most commonly encountered methods.

The choice of group linkage method can dramatically affect dendrogram structure – see below and McCune & Grace (2002; Fig. 11.4) for examples, and play with the data below to see this for yourself. Singh et al. (2011) found that Ward's linkage worked better than average or complete linkage methods. McCune & Grace (2002) recommend using Ward's method, flexible beta with beta = -0.25, or the group average method.

Key Takeaways

The group linkage method chosen can dramatically affect the structure of the resulting dendrogram. Since this isn't a statistical test, it's ok to explore alternatives.

Hierarchical Polythetic Agglomerative Cluster Analysis in R

Hierarchical polythetic agglomerative cluster analysis – illustrated in our simple example above – is a commonly used technique. Is it the best technique? Well, that depends on many things, including how important you consider the different questions that distinguish types of cluster analyses. It is also helpful to recall that cluster analysis is not testing hypotheses. If several clustering methods seem appropriate, Legendre & Legendre (2012) recommend that they all be applied and the results compared. If comparable clusters are identified via several methods, those clusters are likely robust and worth explaining.

There are numerous functions to conduct hierarchical cluster analysis in R. We'll review some of the most common.

stats::hclust()

The `hclust()` function is available in the `stats` package, which is automatically loaded when R is opened. Its usage is:

```
hclust(  
  d,  
  method = "complete",  
  members = NULL  
)
```

The arguments are:

- `d` – the dissimilarity matrix to be analyzed
- `method` – the agglomeration (aka group linkage) method to be used. Choices are:
 - Ward's minimum variance method. In recent versions of R, there are two variations on this

method:

- `ward.D` – this used to be the only option for Ward’s method, and was called ‘ward’ in R until version 3.0.3. However, it is not the version that was originally described by Ward (1963) or recommended by McCune & Grace (2002).
- `ward.D2` – this is the version that Ward (1963) originally described, and is available as an option for this argument in R 3.1.0 and newer. It differs from `ward.D` in that the distances are squared during the algorithm. One of the methods recommended by McCune & Grace (2002).
- `single` – Single linkage.
- `complete` – Complete linkage. The default method.
- `average` – aka UPGMA. One of the methods recommended by McCune & Grace (2002).
- `mcquitty` – aka WPGMA.
- `median` – aka WPGMC.
- `centroid` – aka UPGMC.
- `members` – a vector assigning sample units to clusters. Optional: this is used to construct a dendrogram beginning in the middle of the cluster analysis (i.e., beginning with the clusters already specified by this argument).

The function returns an object of class ‘hclust’ containing the following components (as usual, most are not displayed automatically; you have to call them to see them):

- `merge` – an $n-1 \times 2$ matrix summarizing the order in which clustering occurred. Each row is a step in the process, and indicates the two clusters that were combined at each step (do you see why there are $n-1$ steps?). Negative values indicate single plots (indexed by their row number in the original distance matrix) and positive values indicate clusters formed at an earlier step (row) in the clustering process. For example, ‘-10’ would indicate the tenth plot in the distance matrix, while ‘10’ would indicate the cluster formed in step 10 of the clustering process.
- `height` – the clustering height (dissimilarity value) at which each step in the clustering process occurred.
- `order` – a listing of the sample units, ordered such that a dendrogram of the sample units ordered this way will not have overlapping or crossed branches.
- `labels` – the label for each sample unit. Corresponds to the sample unit (row) names in the original data matrix.
- `method` – the clustering method used.
- `call` – the call which generated this result.
- `dist.method` – the distance method used to create `d`.

Oak Example

Let’s apply this to our oak plant community dataset. Use the `load.oak.data.R` script to load the datafile and make our standard adjustments – removing rare species and relativizing by species maxima. After doing so, the object `oak1.dist` contains the Bray-Curtis distance between the 47 stands as calculated from the relativized abundances of 103 common species.

```
source("scripts/load.oak.data.R")
```

We’ll apply hierarchical clustering to the compositional data, using Ward’s linkage method (`ward.D2`):

```
Oak1.hclust <- hclust(d = Oak1.dist, method = "ward.D2")
```

```
Oak1.hclust
```

```
Call:
hclust(d = Oak1.dist, method = "ward.D2")

Cluster method      : ward.D2
Distance            : bray
Number of objects: 47
```

Displaying the output to the screen doesn't display much useful information, but we can (of course) call components of the output separately:

`Oak1.hclust$height`

```
[1] 0.2687201 0.3130190 0.3752941 0.3991791 0.4249295 0.4332091 0.4412585 0.4459536
[9] 0.4537384 0.4722402 0.4763079 0.4818018 0.4850890 0.4947853 0.4955957 0.5120773
[17] 0.5132889 0.5172461 0.5304432 0.5318053 0.5415588 0.5558036 0.5828645 0.5840160
[25] 0.5849669 0.6125170 0.6163539 0.6191646 0.6442496 0.6461427 0.6673575 0.7111108
[33] 0.7368921 0.7524170 0.7838412 0.7918328 0.8161702 0.8330401 0.8698169 0.8700274
[41] 0.8775783 1.0135263 1.0889640 1.3354662 1.4127213 1.5661029
```

`Oak1.hclust$merge %>%
head()`

```
      [,1] [,2]
[1,]  -22  -24
[2,]   -1   -2
[3,]  -13  -14
[4,]   -3  -20
[5,]  -26  -31
[6,]  -17  -18
```

(note that I only show the first 6 lines here)

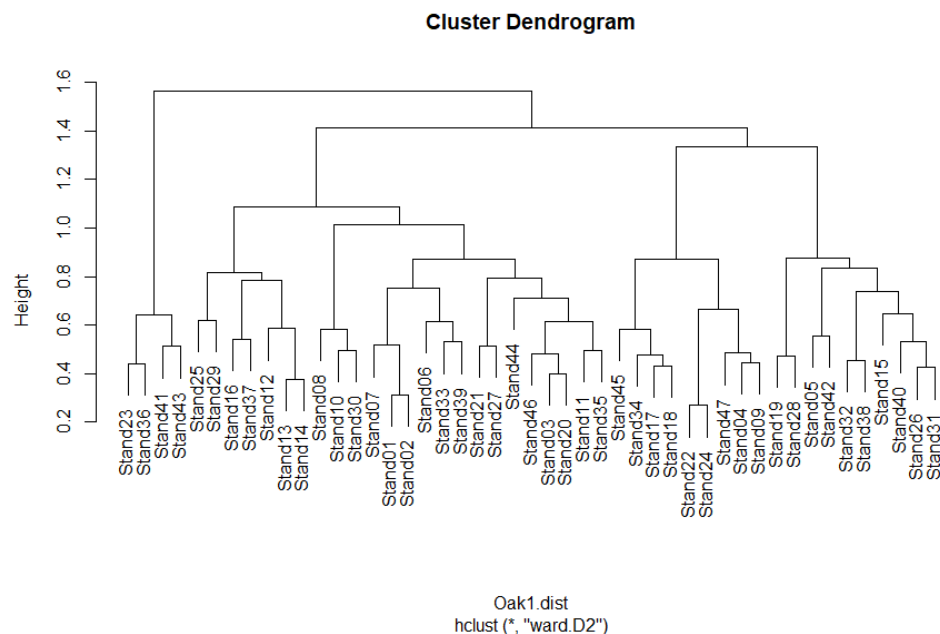
Refer to the description of the output above to see how to interpret these components.

Customizing Dendrograms

One of the attractive elements of hierarchical cluster analysis is its ability to visualize the similarity among observations via a **dendrogram**. A dendrogram is a graphical representation of the `height` and `merge` components from the `hclust()` output.

The `plot()` function automatically creates a dendrogram for objects of class 'hclust':

```
plot(Oak1.hclust)
```



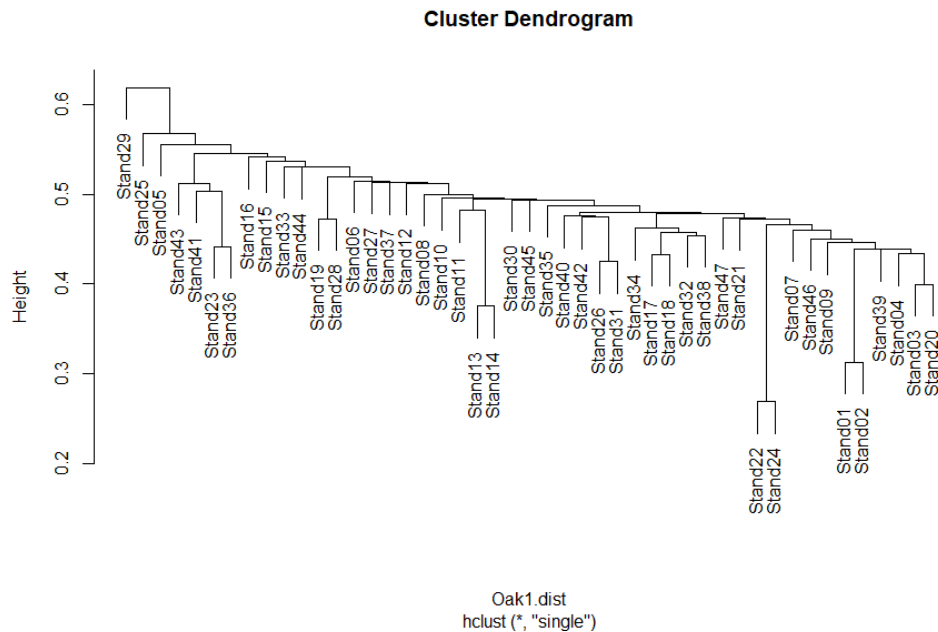
Dendrogram of the oak plant community dataset (47 sample units). Rare species were removed and abundances were relativized by species maxima before being expressed as Bray-Curtis dissimilarity. Ward's linkage method was the group linkage method.

See the interpretive notes for dendrograms in the simple example above. A few follow-up items:

- The dendrogram axis can be scaled in different ways:
 - Fusion distances (termed 'height' in the above dendrogram)
 - Wishart's objective function – the sum of the error sum of squares from each centroid to the items in that group. Not very easily interpreted.
 - % information remaining – a rescaling of Wishart's function. Declines from 100% when each sample unit is its own group to 0% when all sample units are in the same group.
- When an `hclust()` object is plotted, the plotting algorithm orders the branches so that tighter clusters are on the left.

Chaining is relatively common when using the single linkage grouping method with this dataset – try it!

```
hclust(Oak1.dist, method = "single") %>% plot()
```



Dendrogram of the oak plant community data, with all characteristics the same as above except that single linkage was the group linkage method.

The same information went into both dendrograms, but they produced very different structures. Since the goal of cluster analysis is to identify distinct groups, can you see why a lot chaining is unhelpful?

Class 'hclust' Dendrograms

The dendrogram shown above is pretty rudimentary and, of course, can be customized.

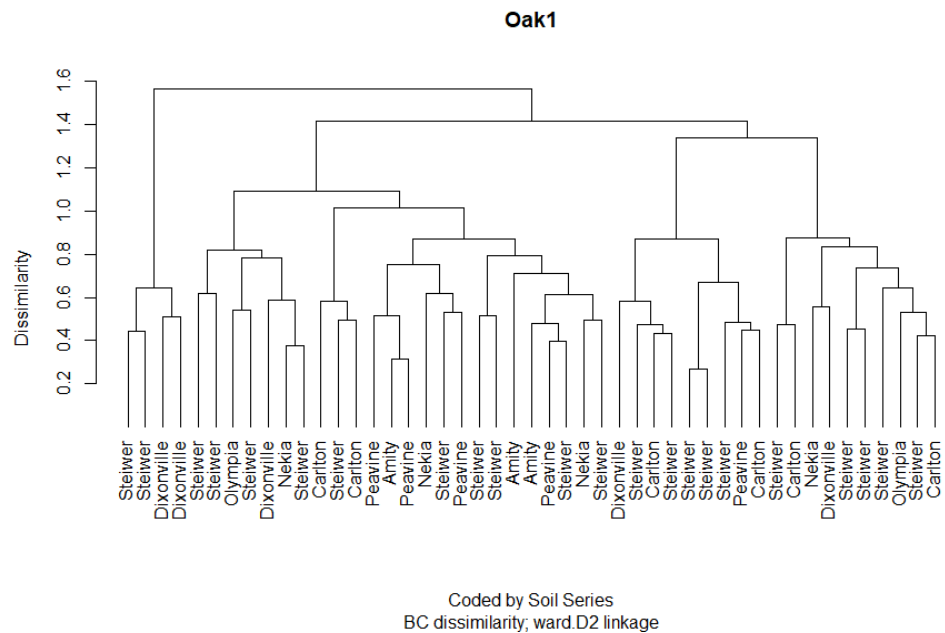
In the base function for plotting objects of class 'hclust', arguments include:

- `hang` – extent to which labels hang below the plot. If negative (`hang = -1`), labels are arranged evenly at the bottom of the dendrogram.
- `labels` – how to identify each sample unit. By default, the sample units are labeled by their row names or row numbers, but another variable can be substituted here. Although these labels can be continuously distributed variables, they're most useful if categorical – can you see why?
- `main` – the title of the dendrogram.
- `ylab` – the caption for the vertical axis.
- `xlab` – the caption for the horizontal axis. The subtitle below this caption can be changed using the `sub` argument; by default it reports the linkage method used.

For example, suppose we were interested in exploring how these groups relate to landform:

```
plot(Oak1.hclust,
     hang = -1,
     main = "Oak1",
```

```
ylab = "Dissimilarity",
xlab = "Coded by Soil Series",
sub = "BC dissimilarity; ward.D2 linkage",
labels = Oak$SoilSeriesName)
```



Dendrogram of the oak plant community dataset, as shown above, but with stand identities replaced by soil series.

Note that we are not testing hypotheses here.

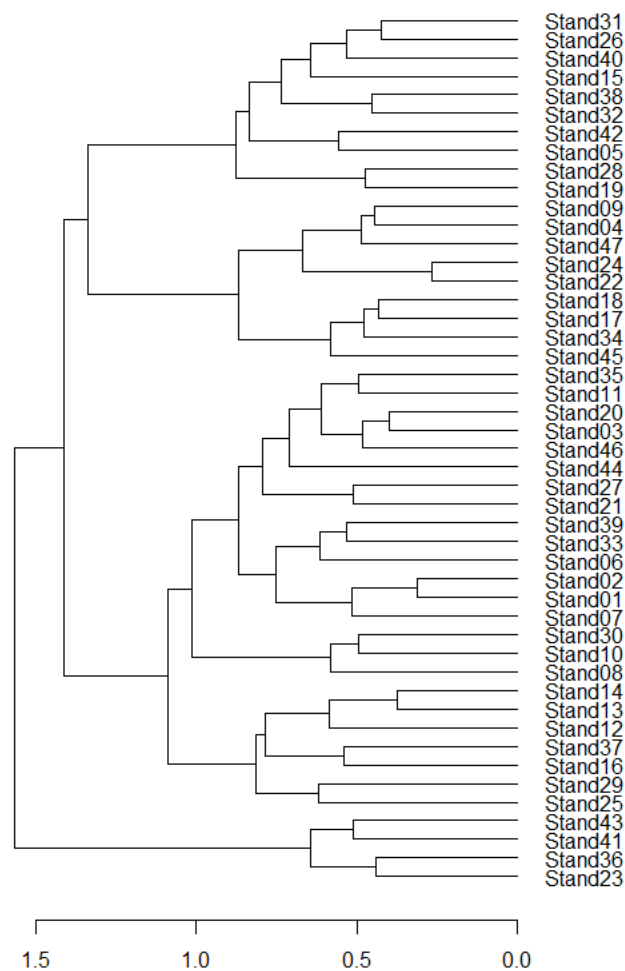
Class 'dendrogram' and the dendextend package

The results of `hclust()` – and other similar functions – can be converted to class 'dendrogram'. This class is intended to be generic and applicable to output from both hierarchical cluster analysis and classification and regression trees. It includes some helpful items such as additional controls on plotting details and the ability to alter the order of sample units in a dendrogram. For more information check the help files (`?as.dendrogram`; `?reorder.dendrogram`).

Applying this to our oak example:

```
Oak1.hclust1 <- as.dendrogram(Oak1.hclust)
```

```
plot(Oak1.hclust1, horiz = TRUE)
```



Dendrogram of the oak plant community data, oriented horizontal rather than vertical.

The `dendextend` package uses objects of class ‘`dendrogram`’ and provides the ability to adjust the graphical parameters of a tree (color, size, type of branches, nodes, leaves, and labels), including creating a dendrogram using `ggplot2`.

Let’s customize our dendrogram a bit:

```
library(tidyverse); library(dendextend)

Oak1.hclust1 %>%
  set("branches_k_color", k = 6) %>%
```

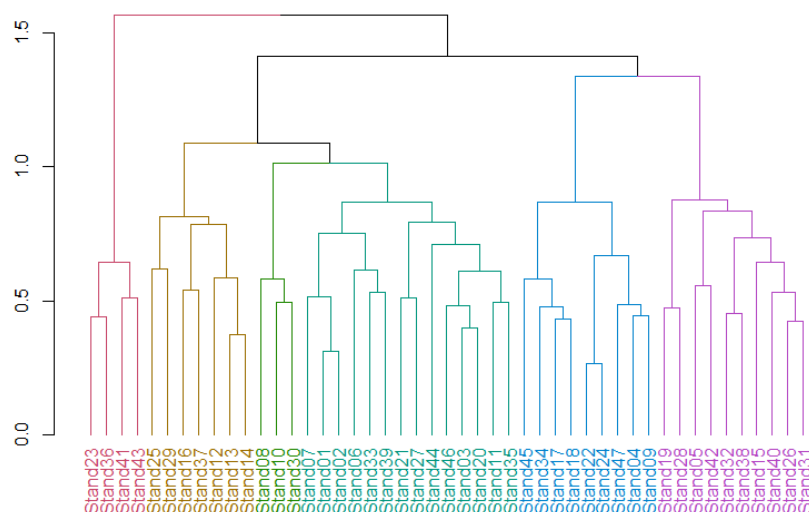


```

set("labels_col", k = 6) %>%
plot

Oak1.hclust1 %>%
rect.dendrogram(k = 6, border = 8, lty = 5, lwd = 1)

```



Dendrogram of the oak plant community data, drawn in dendextend so that branch and label colors distinguish six groups.

In this dendrogram, colors distinguish stands into 6 groups. Each group can be seen by tracing up the dendrogram until it joins with another group. See ‘How Many Groups?’ below for more information about choosing the number of groups.

The customizations that we did here, and many others, are described in the `dendextend` vignette (Galili 2023). The package includes other features such as the `tanglegram()` function which plots two dendrograms against one another, with lines connecting the same observation in each. This permits visual and statistical comparisons of dendrograms.

In addition, `dendextend` can be combined with the `heatmaply` package to combine dendrograms with heat maps (see below for another example).

The ggraph package

The `ggraph` package builds upon `ggplot2`. I haven’t had a chance to explore this in much detail yet, but the associated vignettes include some impressive and highly customized dendrograms (among other types of graphics).

How Many Groups?

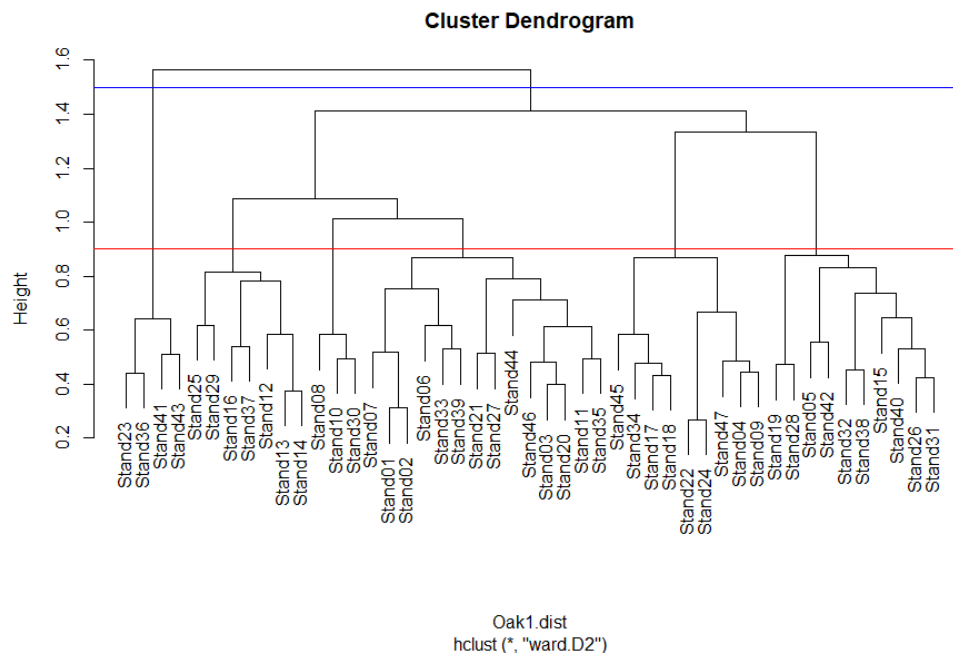
The goal of cluster analysis is to identify discrete groups. The bottom of a dendrogram is not much use – each sample unit is its own group – but the top is not much use either – all sample units are in one group. What we are interested in an intermediate step that classifies the sample units into a reasonable number of groups.

There are many ways to determine how many groups to include in the results of a cluster analysis. **The 'correct' number of groups is a balance between interpretability, sample size, the desired degree of resolution, and other factors.** Too many groups will be difficult to explain (and is counter to the objective of a cluster analysis), whereas too few groups may result in groups that are too heterogeneous to be easily explained.

Exploring the Dendrogram

One way to visualize the number of groups at a given level of the dendrogram is to extend a line perpendicular to the dendrogram axis at that level. Every segment the line encounters is a group.

```
plot(Oak1.hclust)
abline(h = 0.9, col = "red") # 6 groups
abline(h = 1.5, col = "blue") # 2 groups
```



Dendrogram of the oak plant community data, with red and blue horizontal lines extending from the axis through the dendrogram. All branches of the dendrogram that intersect with each colored line would be a group at that height on the axis. Here, 2 groups are shown (blue line) and 6 groups are shown (red line).

The `identify()` function enables you to click on the dendrogram and see which sample units are in a given group:

```
identify(Oak1.hclust)
```

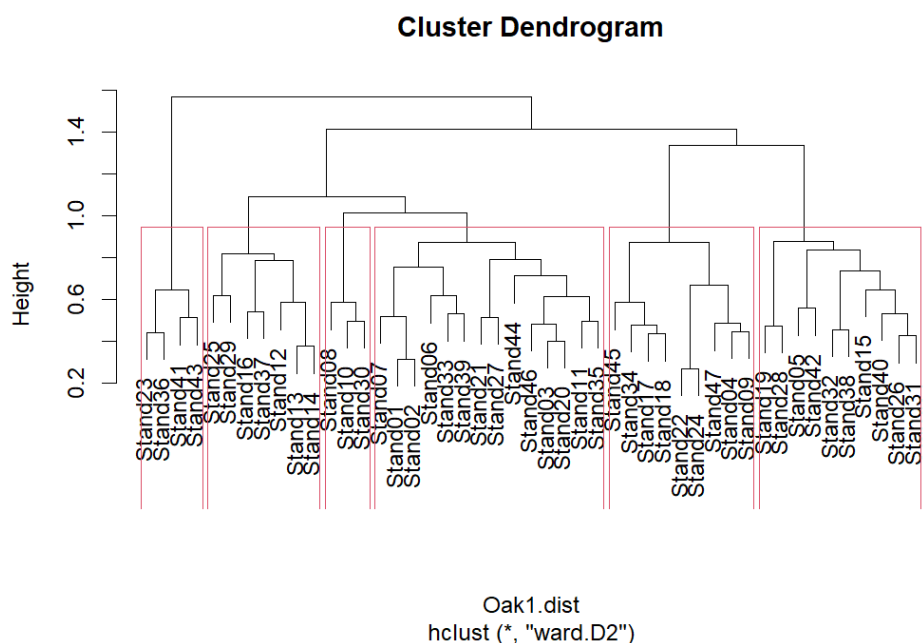
To exit this function, press the 'escape' button or select 'Finish' in the graphing window.

If the output of the `identify()` function is assigned to a new object, a list is produced of which samples are in each of the selected groups. However, this requires manual input and is therefore not easily scriptable.

The `rect.hclust()` function automatically identifies which sample units are part of each group at a given level of the dendrogram. It requires that you specify how many groups (`k`) you want to identify. If this function is implemented when a dendrogram is active in the R Graphics window, the rectangles identifying the groups are simultaneously drawn on the dendrogram:

```
plot(Oak1.hclust)
```

```
group.6 <- rect.hclust(Oak1.hclust, k = 6)
```



Dendrogram of the oak plant community data, with boxes delineating 6 groups.

Viewing this object:

```
group.6
```

```
[[1]]  
Stand23 Stand36 Stand41 Stand43  
  23      36      41      43
```

```

[[2]]
Stand12 Stand13 Stand14 Stand16 Stand25 Stand29 Stand37
      12      13      14      16      25      29      37

[[3]]
Stand08 Stand10 Stand30
      8      10      30

[[4]]
Stand01 Stand02 Stand03 Stand06 Stand07 Stand11 Stand20 Stand21 Stand27 Stand33 Stand35
      1       2       3       6       7      11      20      21      27      33      35
Stand39 Stand44 Stand46
      39      44      46

[[5]]
Stand04 Stand09 Stand17 Stand18 Stand22 Stand24 Stand34 Stand45 Stand47
      4       9      17      18      22      24      34      45      47

[[6]]
Stand05 Stand15 Stand19 Stand26 Stand28 Stand31 Stand32 Stand38 Stand40 Stand42
      5      15      19      26      28      31      32      38      40      42

```

Each group is 'named' by an integer, proceeding from left to right across the dendrogram. However, keep in mind that the ordering of these groups is of little utility. The most important thing is to track which sample units are classified to the same group, not the number that group is assigned to.

Scree Plots

Fusion distances can be plotted against the number of clusters to see if there are sharp changes in the scree plot. If so, the preferred number of groups is the number at the elbow of the scree plot. A smaller number of groups will leave considerable variation unexplained, while a larger number of groups will not explain that much more variation than is explained at the elbow.

The required data are available in the merge and height components returned by `hclust()`. Since we are conducting agglomerative (bottom up) clustering, the last heights represent the last fusions in the dendrogram. We can combine the height and merge components and then plot the last few fusions. Rather than having to re-code this each time, here is a simple function to do so:

```

scree <- function(hclust.obj, max.size = 12) {
  temp <- with(hclust.obj, cbind(height, merge))
  colnames(temp) <- c("Height", "JoinsThis", "WithThis")
  plot(x = max.size:1,
       y = temp[(nrow(temp)-( max.size -1)):nrow(temp), 1],
       xlab = "Number of groups", ylab = "Height",
       main = hclust.obj $call, type = "b")
  tail(temp, max.size)
}

```

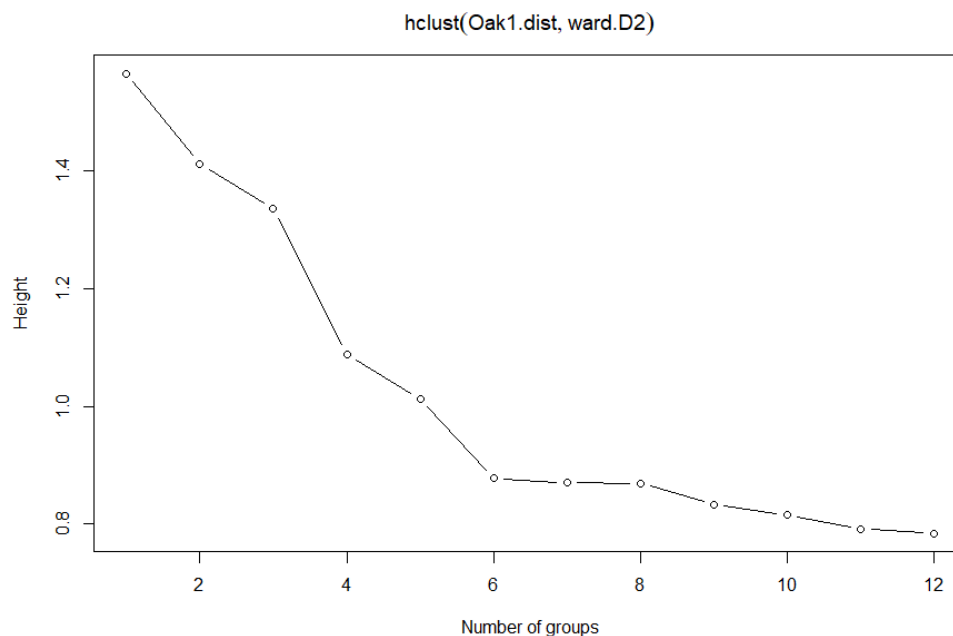
This function is available in the 'functions' folder in the GitHub repository. It has two arguments:

- `hclust.obj` – the object produced by the `hclust()` function.
- `max.size` – how many fusions to plot. Default is the last 12.

The arguments are shown in red font in the above code to highlight where they are used.

After downloading this function to the 'functions' subfolder in your working directory, load and then apply it:

```
source("functions/scree.R")  
scree(hclust.obj = Oak1.hclust)
```



Scree plot of a hierarchical cluster analysis of the oak plant community data using Ward's group linkage method.

Which number of groups appears to be optimal?

stats::cutree()

A dendrogram can be cut into groups using the `cutree()` function. The usage is:

```
cutree(  
  tree,  
  k = NULL,  
  h = NULL  
)
```

The arguments are:

- `tree` – results from a `hclust()` function.
- `k` – desired number of groups. Can be a scalar (e.g., 2 groups) or a vector (e.g., 2:7 groups).
- `h` – height(s) where tree should be cut.

You need to specify `tree` and either `k` or `h`.

```
g6 <- cutree(Oak1.hclust, k = 6); g6
```

Stand01	Stand02	Stand03	Stand04	Stand05	Stand06	Stand07	Stand08	Stand09	Stand10	Stand11
1	1	1	2	3	1	1	4	2	4	1
Stand12	Stand13	Stand14	Stand15	Stand16	Stand17	Stand18	Stand19	Stand20	Stand21	Stand22
5	5	5	3	5	2	2	3	1	1	2
Stand23	Stand24	Stand25	Stand26	Stand27	Stand28	Stand29	Stand30	Stand31	Stand32	Stand33
6	2	5	3	1	3	5	4	3	3	1
Stand34	Stand35	Stand36	Stand37	Stand38	Stand39	Stand40	Stand41	Stand42	Stand43	Stand44
2	1	6	5	3	1	3	6	3	6	1
Stand45	Stand46	Stand47								
2	1	2								

This information can also be easily summarized:

```
table(g6)
```

```
g6
 1  2  3  4  5  6
14 9 10 3  7  4
```

The `cutree()` function can also be used to explore various numbers of groups by varying `k`:

```
cutree(Oak1.hclust, k = 2:7) %>%
  head() # Why isn't k=1 included here?
```

	2	3	4	5	6	7
Stand01	1	1	1	1	1	1
Stand02	1	1	1	1	1	1
Stand03	1	1	1	1	1	1
Stand04	1	2	2	2	2	2
Stand05	1	2	3	3	3	3
Stand06	1	1	1	1	1	1

This function is similar to `rect.hclust()` but produces an object of a different class. Each column refers to a different number of groups, and the integers within the column identify which observations are assigned to the same group. I included the `head()` function above to reduce the output for visual purposes; I obviously would not want to do so if saving the output to an object.

Confusion Matrices

Finally, the results of using different values of `k`, or different clustering methods, can be compared in

a **confusion matrix**. For example, compare the observations assigned to clusters assigned to 4 or 6 groups:

```
table(g6, cutree(Oak1.hclust, k = 4))
```

g6	1	2	3	4
1	14	0	0	0
2	0	9	0	0
3	0	0	10	0
4	3	0	0	0
5	7	0	0	0
6	0	0	0	4

The confusion matrix illustrates the hierarchical nature of this approach – observations that are grouped together earlier in the process are kept together in all subsequent steps. For example, group 1 from the 4-group solution consists of all observations classified to groups 1, 4, and 5 in the 6-group solution.

Other Approaches For Selecting Groups

Other approaches have also been suggested for deciding how many groups to retain:

- McKenna (2003) outlined a method to provide ‘objective significance determination’ of the groups identified in a cluster analysis.
- Indicator Species Analysis (ISA) was originally proposed as a means for deciding how many groups to include in a cluster analysis (Dufrêne & Legendre 1997). ISA can also be used for other purposes; we'll consider it separately.
- If you have independent data for your sample units (e.g., environmental variables), you can use these data to examine whether the groups make sense. For example, do environmental variables differ among the groups identified through the cluster analysis? This is what McCune et al (2000) did in their cluster analysis of epiphyte species (see their Fig. 3), retaining about 45% of the information and 9 groups.
- Significance can be assessed via bootstrapping using the `pvc1ust()` function in the `pvc1ust` package (Suzuki & Shimodaira 2006).
- The `progenyClust` package has a method for identifying the optimal number of clusters (Hu & Qutub 2016).
- The `NbClust` package provides 30 different indices to determine the number of clusters in a dataset (Charrad et al. 2014). Of particular interest is the ‘Gap statistic’, which compares the intra-cluster variation for different values of k to a distribution of variation values obtained by Monte Carlo simulations (Tibshirani et al. 2001). The recommended number of clusters (k) is the smallest value of k with a Gap statistic within one standard deviation of the Gap statistic at $k+1$. This approach can be applied to any clustering method, whether hierarchical or non-hierarchical. For the oak plant community data:

```
require(NbClust)
```

```
NbClust(data = Oak1, diss = Oak1.dist, distance = NULL, method = "ward.D2",  
index = "gap")
```

How many groups does this approach suggest for this dataset?

Other Types of Clustering

Other hierarchical clustering techniques have also been developed. Two examples are:

- **Fuzzy clustering:** All of the variations we've listed here classify sample units as being in group A or B, but another way to look at this is in terms of the probability that a given sample unit belongs in each group. For example, some sample units might clearly belong to one group whereas others are marginal – they could just as easily be assigned to group B as to group A. For more information, see Equihua (1990) and Boyce & Ellison (2001). One function for fuzzy clustering is `cluster::fanny()`.
- **Spatially-constrained clustering:** When explicit spatial information is available, it can be used to group objects that are near one another and similar. For example, this approach has been used to identify clusters of trees in a mapped forest stand. For more information, see Plotkin et al. (2002). Churchill et al. (2013) used this approach to identify clumps of trees and thereby guide the thinning of forest stands to restore spatial patterning.

Conclusions

It is worth repeating that a cluster analysis does not test hypotheses. It is acceptable to explore various distance measures, clustering techniques, numbers of groups, etc. to identify a preferred solution.

Note that we've used a cluster analysis to group sample units with similar compositions. However, we also could have transposed the data matrix and used a cluster analysis to group species that co-occur. See McCune & Grace (2002, p.90) for a description of the considerations when doing so, specifically with respect to relativization.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Boyce, R.L., and P.C. Ellison. 2001. Choosing the best similarity index when performing fuzzy set ordination on binary data. *Journal of Vegetation Science* 12:711-720.
- Charrad, M., N. Ghazzali, V. Boiteau, and A. Niknafs. 2014. NbClust: an R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software* 61(6):1-36.
- Churchill, D.J., A.J. Larson, M.C. Dahlgreen, J.F. Franklin, P.F. Hessburg, and J.A. Lutz. 2013. Restoring forest resilience: from reference spatial patterns to silvicultural prescriptions and monitoring. *Forest Ecology and Management* 291:442-457.
- Dufrêne, M., and P. Legendre. 1997. Species assemblages and indicator species: the need for a flexible asymmetrical approach. *Ecological Monographs* 67:345-366.
- Equihua, M. 1990. Fuzzy clustering of ecological data. *Journal of Ecology* 78:519-534.
- Galili, T. 2023. Introduction to dendextend. <https://cran.r-project.org/web/packages/dendextend/vignettes/dendextend.html>

- Hu, C.W., and A.A. Qutub. 2016. progenyClust: an R package for progeny clustering. *The R Journal* 8:328-338.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- McCune, B., R. Rosentreter, J.M. Ponzetti, and D.C. Shaw. 2000. Epiphyte habitats in an old conifer forest in western Washington, U.S.A. *The Bryologist* 103:417-427.
- McKenna Jr., J.E. 2003. An enhanced cluster analysis program with bootstrap significance testing for ecological community analysis. *Environmental Modelling & Software* 18:205-220.
- Plotkin, J.B., J. Chave, and P.S. Ashton. 2002. Cluster analysis of spatial patterns in Malaysian tree species. *American Naturalist* 160:629-644.
- Singh, W., E. Hjorleifsson, and G. Stefansson. 2011. Robustness of fish assemblages derived from three hierarchical agglomerative clustering algorithms performed on Icelandic groundfish survey data. *ICES Journal of Marine Science* 68:189-200.
- Suzuki, R., and H. Shimodaira. 2006. Pvclust: an R package for assessing the uncertainty in hierarchical clustering. *Bioinformatics* 22:1540-1542.
- Tibshirani, R., G. Walther, and T. Hastie. 2001. Estimating the number of data clusters via the Gap statistic. *Journal of the Royal Statistical Society B* 63:411-423.
- Ward Jr., J.H. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association* 58:236-244.

Media Attributions

- cophenetic
- dendrogram
- Oak.hclust
- oak.hclust.single
- oak.hclust.soilseries
- oak.hclust1.horizontal
- oak.hclust.dendextend
- oak.hclust.2v6groups
- oak.dendrogram5
- oak.scree

29. k-Means Cluster Analysis

Learning Objectives

To consider ways of classifying sample units into non-hierarchical groups, including decisions about how many groups are desired and which criterion to use when doing so.

To begin visualizing patterns among sample units.

Key Packages

```
require(stats, cluster)
```

Introduction

k-means cluster analysis is a non-hierarchical technique. It seeks to partition the sample units into *k* groups in a way that minimizes some criterion. Often, the criterion relates to the variance between points and the centroid of the groups they are assigned to.

An essential part of a *k*-means cluster analysis, of course, is the decision of how many clusters to include in the solution. One way to approach this is to conduct multiple analyses for different values of *k* and to then compare those analyses. For example, Everitt & Hothorn (2006) provide some nice code to compare the within-group sums of squares against *k*. Another approach is to use the results of a hierarchical clustering method as the starting values for a *k*-means analysis. One of the older posts on Stack Overflow contains many examples of how to determine an appropriate number of clusters to use in a *k*-means cluster analysis. Additional ideas about how to decide how many clusters to use are provided below.

The criterion in *k*-means clustering is generally to minimize the variance within groups, but how variance is calculated varies among algorithms. For example, `kmeans()` seeks to minimize within-group sums of squared distances. Another function, `pam()`, seeks to minimize within-group sums of dissimilarities, and can be applied to any distance matrix. Both functions are described below.

***k*-means cluster analysis is an iterative process:**

- Select *k* starting locations (centroids)
- Assign each observation to the group to which it is closest
- Calculate within-group sums of squares (or other criterion)

- Adjust coordinates of the k locations to reduce variance
- Re-assign observations to groups, re-assess variance
- Repeat process until stopping criterion is met

The iterative nature of this technique can be seen using the `animation::kmeans.ani()` function (see the examples in the help files) or in videos like this: <https://www.youtube.com/watch?v=BVFG7fd1H30>. We will see a similar iterative process with NMDS.

k -means clustering may find a local (rather than global) minimum value due to differences in starting locations. A common solution to this is to use multiple starts, thus increasing the likelihood of detecting a global minimum. This, again, is similar to the approach taken in NMDS.

As usual, we will illustrate these approaches with our oak dataset.

```
source("scripts/load.oak.data.R")
```

k -means Clustering in R

In R, k -means cluster analysis can be conducted using a variety of methods, including `stats::kmeans()`, `cluster::pam()`, `cluster::clara()`, and `cluster::fanny()`.

stats::kmeans()

The `kmeans()` function is available in the `stats` package. It seeks to identify the solution that minimizes the sum of squared (Euclidean) distances. Its usage is:

```
kmeans(x,
  centers,
  iter.max = 10,
  nstart = 1,
  algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
  trace = FALSE
)
```

The key arguments are:

- `x` – data matrix. Note that a distance matrix is not allowed. Euclidean distances are assumed to be appropriate.
- `centers` – the number of clusters to be used. This is the ‘ k ’ in k -means.
- `iter.max` – maximum number of iterations allowed. Default is 10.
- `nstart` – number of random sets of centers to choose. Default is 1.
- `algorithm` – the algorithm used to derive the solution. I won’t pretend to understand what these algorithms are or how they differ. 😊 The default is “Hartigan-Wong”.

This function returns an object of class ‘`kmeans`’ with the following values:

- `cluster` – a vector indicating which cluster each sample unit is assigned to. Note that the coding of clusters is arbitrary, and that different runs of the function may result in different

coding.

- `centers` – a matrix showing the mean value of each species in each sample unit.
- `totss` – the total sum of squares.
- `withinss` – the within-cluster sum of squares for each cluster.
- `tot.withinss` – the sum of the within-cluster sum of squares
- `betweenss` – the between-cluster sum of squares.
- `size` – the number of sample units in each cluster.
- `iter` – the number of iterations

An example for six groups:

```
k6 <- kmeans(x = Oak1, centers = 6, nstart = 100)
```

Note that we did not make any adjustments to our data but are using this here to illustrate the technique. Also, your results may differ depending on the number of starts (`nstart`) and the number of iterations (`iter.max`). In addition, your groups may be identified by different integers. To see the group assignment of each sample unit:

```
k6$cluster
```

```
Stand01 Stand02 Stand03 Stand04 Stand05 Stand06 Stand07
      1      1      1      5      4      2      1
Stand08 Stand09 Stand10 Stand11 Stand12 Stand13 Stand14
      5      5      6      1      1      1      1
Stand15 Stand16 Stand17 Stand18 Stand19 Stand20 Stand21
      4      6      5      5      4      1      1
Stand22 Stand23 Stand24 Stand25 Stand26 Stand27 Stand28
      5      3      5      1      4      1      4
Stand29 Stand30 Stand31 Stand32 Stand33 Stand34 Stand35
      4      5      4      4      2      5      1
Stand36 Stand37 Stand38 Stand39 Stand40 Stand41 Stand42
      3      4      4      1      4      2      4
Stand43 Stand44 Stand45 Stand46 Stand47
      2      6      5      5      5
```

A more compact summary:

```
table(k6$cluster)
```

```
 1  2  3  4  5  6
 2 12 12 14  3  4
```

vegan::cascadeKM()

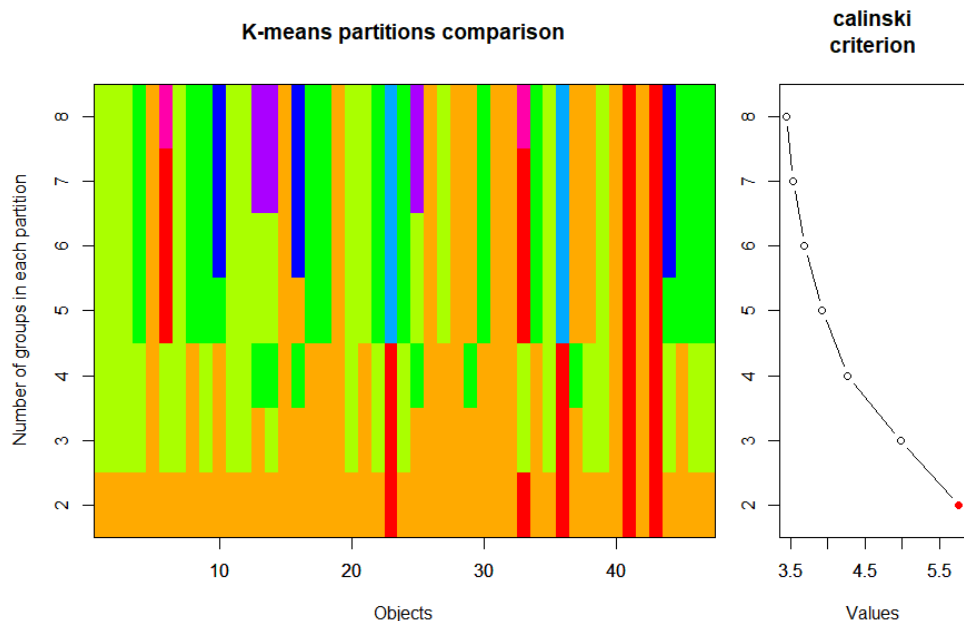
The `kmeans()` function permits a cluster analysis for one value of k . The `cascadeKM()` function, available in the `vegan` package, conducts multiple `kmeans()` analyses: one for each value of k ranging from the smallest number of groups (`inf.gr`) to the largest number of groups (`sup.gr`):

```
cascadeKM(Oak1, inf.gr = 2, sup.gr = 8)
```

(output not displayed in notes)

The output can also be plotted:

```
plot(cascadeKM(Oak1, inf.gr = 2, sup.gr = 8))
```



K-means cluster analysis of oak plant community data for $k=2$ through $k=8$ groups (vertical axis). Stands are shown on the horizontal axis. Within each row, colors distinguish different groups. On the left is the Calinski criterion – larger values indicate more support for that group size.

It is important to recall that k -means solutions are not hierarchical – you can see evidence of this by looking at the colors in this graph.

cluster::pam()

The `pam()` function is available in the `cluster` package. The name is an abbreviation for ‘partitioning around medoids’ (a medoid is a representative example of a group – in this case, the sample unit that is most similar to all other sample units in the cluster). It is more flexible than `kmeans()` because it allows the use of distance matrices produced from non-Euclidean measures. It also uses a different criterion, identifying the solution that minimizes the sum of dissimilarities. Its usage is:

```
pam(x,  
    k,  
    diss = inherits(x, "dist"),  
    metric = c("euclidean", "manhattan"),  
    medoids = if (is.numeric(nstart)) "random",  
    nstart = if (variant == "faster") 1 else NA,  
    stand = FALSE,
```

```

cluster.only = FALSE,
do.swap = TRUE,
keep.diss = !diss && !cluster.only && n < 100,
keep.data = !diss && !cluster.only,
variant = c("original", "o_1", "o_2", "f_3", "f_4", "f_5", "faster"),
pamonce = FALSE,
trace.lev = 0
)

```

There are quite a few arguments in this function, but the key ones are:

- **x** – data matrix or data frame, or distance matrix. Since this accepts a distance matrix as an input, it does not assume Euclidean distances as is the case with `kmeans()`.
- **k** – the number of clusters to be used
- **diss** – logical; **x** is treated as a distance matrix if **TRUE** (default) and as a data matrix if **FALSE**.
- **metric** – name of distance measure to apply to data matrix or data frame. Options are “euclidean” and “manhattan”. Ignored if **x** is a distance matrix.
- **medoids** – if you want specific sample units to be the medoids, you can specify them with this argument. The default (**NULL**) results in the medoids being identified by the algorithm.
- **stand** – whether to normalize the data (i.e., subtract mean and divide by SD for each column). Default is to not do so (**FALSE**). Ignored if **x** is a dissimilarity matrix.

This function returns an object of class ‘pam’ that includes the following values:

- **medoids** – the identity of the sample units identified as ‘representative’ for each cluster.
- **clustering** – a vector indicating which cluster each sample unit is assigned to.
- **silinfo** – silhouette width information. The silhouette width of an observation compares the average dissimilarity between it and the other points in the cluster to which it is assigned and between it and points in other clusters. A larger value indicates stronger separation or strong agreement with the cluster assignment.

An example for six groups:

```
k6.pam <- pam(x = Oak1.dist, k = 6)
```

Note that we called `Oak1.dist`, which is based on Bray-Curtis dissimilarities, directly since `pam()` cannot calculate this measure internally. Your groups may be identified by different integers. To see the group assignment of each sample unit:

```
k6.pam$clustering
```

Stand01	Stand02	Stand03	Stand04	Stand05	Stand06	Stand07
1	1	1	2	3	1	1
Stand08	Stand09	Stand10	Stand11	Stand12	Stand13	Stand14
2	2	4	2	1	5	5
Stand15	Stand16	Stand17	Stand18	Stand19	Stand20	Stand21
4	5	3	3	3	2	3
Stand22	Stand23	Stand24	Stand25	Stand26	Stand27	Stand28
2	6	2	5	4	3	4
Stand29	Stand30	Stand31	Stand32	Stand33	Stand34	Stand35
5	4	4	3	6	3	2

Stand36	Stand37	Stand38	Stand39	Stand40	Stand41	Stand42
6	4	2	4	4	6	2
Stand43	Stand44	Stand45	Stand46	Stand47		
6	2	2	2	2		

The integers associated with each stand indicate the group to which it is assigned.

The help file about the object that is produced by this analysis (`?pam.object`) includes an example in which different numbers of clusters are tested systematically and the size with the smallest average silhouette width is selected as the best number of clusters. Here, I've modified this code for our oak plant community dataset:

```
asw <- numeric(10)
for (k in 2:10) {
  asw[k] <- pam(Oak1.dist, k = k) $ silinfo $ avg.width
}
cat("silhouette-optimal # of clusters:", which.max(asw), "\n")
```

```
silhouette-optimal # of clusters: 2
```

This algorithm suggests that there is most support for two clusters of sample units. We could re-run `pam()` with this specific number of clusters if desired.

Comparing *k*-means Analyses

Many of the techniques that we saw previously for comparing hierarchical cluster analyses are also relevant here. For example, we could generate a confusion matrix comparing the classifications from the `kmeans()` and `pam()`.

```
table(k6$cluster, k6.pam$clustering)
```

	1	2	3	4	5	6
1	5	3	2	1	3	0
2	1	0	0	0	0	3
3	0	8	3	1	0	0
4	0	0	0	0	0	2
5	0	1	0	1	1	0
6	0	2	3	6	1	0

This can be a bit confusing to read because the row and column names are the integers 1:6 but the cells within the table are also numbers (i.e., number of stands in each group). Also, the number of groups is the same in both analyses so it isn't immediately apparent which analysis is shown as rows and which as columns.

Note that observations are more easily assigned to different groups in *k*-means analyses than they are in hierarchical analysis: in the latter, once two sample units are fused together they remain fused through the rest of the analysis.

Conclusions

k-means cluster analysis is a technique for classifying sample units into *k* groups. This approach is not constrained by the decisions that might have been made earlier in a hierarchical analysis. There is also no concern about other factors such as which group linkage method to use.

However, *k*-means cluster analysis does require that the user decide how many groups (*k*) to focus on. In addition, since the process is iterative and starts from random coordinates, it's possible to end up with different classifications from one run to another. Finally, functions such as `kmeans()` use a Euclidean criterion and thus are most appropriate for data expressed with that distance measure. To apply this technique to a semimetric measure such as Bray-Curtis dissimilarity, some authors have suggested standardizing data using a Hellinger transformation: the square root of the data divided by the row total. This standardization is available in `decostand()`, but I have not explored its utility.

References

Everitt, B.S., and T. Hothorn. 2006. *A handbook of statistical analyses using R*. Chapman & Hall/CRC, Boca Raton, LA.

Media Attributions

- oak.cascadeKM

30. Using Groups

Learning Objectives

To explore ways of using the groups identified from cluster analyses: naming them, using them to summarize data, and overlaying them onto an ordination.

To continue visualizing patterns among sample units.

Key Packages

```
require(vegan)
```

Introduction

A hierarchical cluster analysis produces a dendrogram showing the relationships among sample units. Using the criteria discussed in that chapter, the analyst decides which height of the dendrogram to focus on and the associated number of groups. Each sample unit is assigned to one group based on where it was fused within the dendrogram.

A non-hierarchical cluster analysis (e.g., *k*-means) also assigns each sample unit to a group. The analyst decides how many groups they want and which criterion to use. Each sample unit is assigned to one group so as to minimize the criterion.

In this chapter, we will illustrate several ways to use the groups identified by either approach.

Oak Example, and Naming Groups

The ideas in this chapter are illustrated with hierarchical and *k*-means cluster analyses of our oak dataset. I'll use 6 groups for each analysis for simplicity.

```
source("scripts/load.oak.data.R")  
  
Oak_explan$Stand <- rownames(Oak)  
  
Oak1.hclust <- hclust(d = Oak1.dist, method = "ward.D2")
```

```
g6 <- cutree(Oak1.hclust, k = 6)

k6 <- kmeans(x = Oak1, centers = 6, nstart = 100)$cluster

Oak_explan <- data.frame(Oak_explan, g6, k6) #Combine grouping codes with original
data
```

It can be confusing to keep track of the groups from different analyses when they're named with the same values (here, 1:6). Also, these numbers are subjective – there's no reason to expect group 1 from the `hclust()` analysis to have any relation to group 1 from the `kmeans()` analysis. Instead, let's give them distinct categorical names.

```
groups <- data.frame(g6, k6) %>%
  rownames_to_column("Stand") %>%
  merge(y = data.frame(g6 = 1:6, g6_letter = letters[1:6])) %>%
  merge(y = data.frame(k6 = 1:6, k6_LETTER = LETTERS[21:26]))

with(groups, table(g6_letter, k6_LETTER))
```

	k6_LETTER					
g6_letter	U	V	W	X	Y	Z
a	10	0	1	2	1	0
b	0	0	9	0	0	0
c	0	0	0	0	0	10
d	0	0	2	0	1	0
e	4	0	0	0	1	2
f	0	2	0	2	0	0

I've highlighted in bold the row and column names – the rows reflect the groups from the `hclust()` analysis and the columns reflect the groups from the `kmeans()` analysis. The values within the table are the number of sample units assigned to the same combination of groupings.

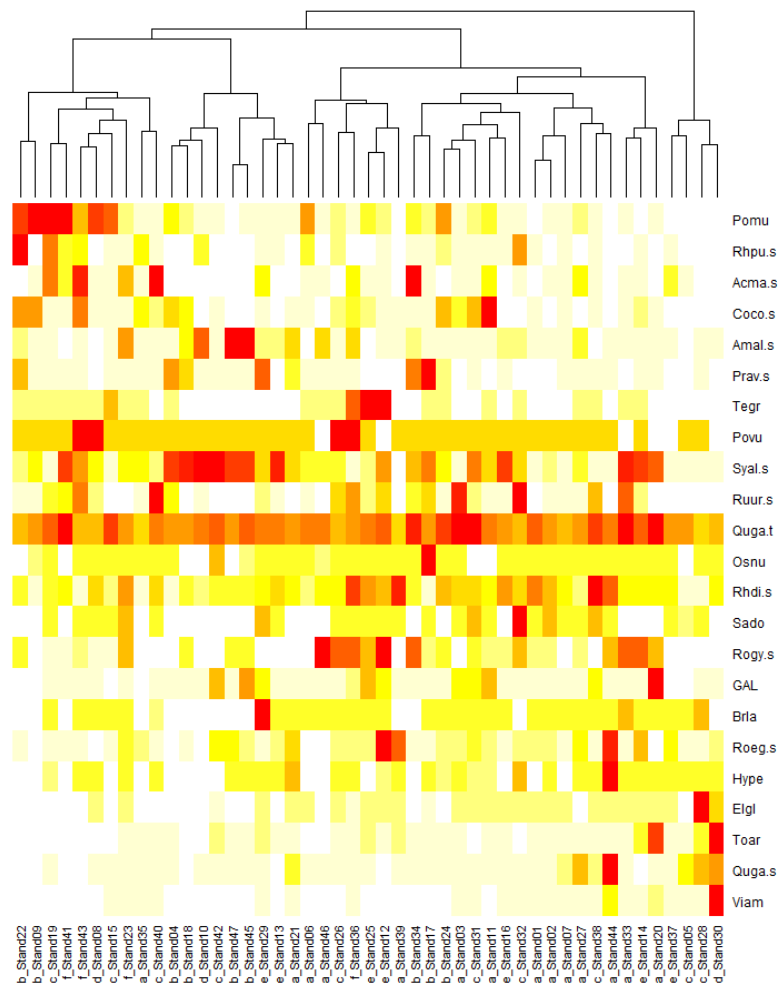
The differing criteria for identifying groups have resulted in some differences between these two approaches – for example, sample units that are assigned to group 'a' in the `hclust()` analysis are assigned to four different groups in the `kmeans()` analysis. Furthermore, the assignment of groups of sample units to particular group codes may vary from run to run of the analysis.

Comparing Dendrogram to Individual Variables

One way to explore a hierarchical cluster analysis is to compare it with the abundances of the species present in each plot.

The `vegan::tabasco()` function allows this. The following graphic would be very busy if I used all species, so I focus just on those species present in at least half of the stands. I also label each sample unit with the group it is assigned to and its stand number.

```
tabasco(x = vegtab(Oak1,
  minval = nrow(Oak1)/2),
  use = Oak1.hclust,
  labCol =paste(groups$g6_letter, groups$Stand, sep = "_") )
```



Heat map showing relative abundance of most common species in oak plant community dataset (rows) within each stand (column), along with a dendrogram from a hierarchical cluster analysis of these data. Red colors indicate high relative abundance, and white indicates absence.

In this image, sample units are arranged based on their location in the dendrogram shown at the top of the heat map and are coded at the bottom of the image by the numerical code for the group to which they are assigned and their stand number.

Each cell in this image is colored to reflect the abundance of that species (row) in that sample unit (column). Note that we relativized each species by its maximum and therefore every row has at least one dark red cell (relative abundance = 1). If we had not relativized these data, the species with larger absolute abundances would be darker in this heat map. As it is, you can see that Quga.t has mostly red colors because it had relatively high abundance in all plots.

The `tabasco()` function can also be applied to view species abundances relative to a vector of other data (e.g., values of an explanatory variable).

The `vegan::vegemite()` function produces a compact tabular summary of the abundances of taxa. It requires specification of a scale for expressing abundance as a set of one-character numbers or symbols.

Summarizing Variables Within Groups

We can treat the groups from a cluster analysis as a factor that is then used to summarize individual variables. These individual variables could be part of the data matrix from which the distance matrix was calculated, or other variables that were collected on the same sample units but were not part of the cluster analysis.

To illustrate this, I use the 6 groups from the hierarchical cluster analysis to summarize a few explanatory variables.

```
Oak_explan %>%
  merge(y = groups) %>%
  group_by(g6_letter) %>%
  summarize(n = length(g6_letter),
    Elev.m = mean(Elev.m),
    AHoriz = mean(AHoriz))
```

```
# A tibble: 6 × 4
  g6_letter      n Elev.m AHoriz
<chr>      <int> <dbl> <dbl>
1 a             14  133.   13.1
2 b              9  135.   25.2
3 c             10  162.    18
4 d              3  157.   11.3
5 e              7  145.   16.9
6 f              4  163.   16.8
```

For example, groups 'c' and 'f' have higher average elevations than the others.

We could also test whether groups differ with regard to elevation and other variables.

We can do the same calculations with response variables. Furthermore, since we relativized each species by its maxima, we can summarize either the raw data or the relativized data. Here's we'll just pick out two species.

Looking at the raw abundance data for these species within the 6 groups from the hierarchical cluster analysis:

```
Oak %>%
  rownames_to_column("Stand") %>%
  merge(y = groups) %>%
  group_by(g6_letter) %>%
  summarize(n = length(g6_letter),
    Pomu = mean(Pomu),
    Syal.s = mean(Syal.s))
```

```
# A tibble: 6 × 4
  g6_letter      n Pomu Syal.s
<chr>      <int> <dbl> <dbl>
1 a             14  8.5  21.3
```

2	b	9	5.58	55.7
3	c	10	48.2	21.2
4	d	3	1.40	57
5	e	7	12.4	14.1
6	f	4	0	0.975

Pomu had much higher cover (48.2%) in group 'c' than in the groups, and was absent from plots classified into group 'f'. Syal.s was abundant in groups 'b' and 'd'.

Looking at the relativized data, as used in the hierarchical cluster analysis:

```
Oak1 %>%
  rownames_to_column("Stand") %>%
  merge(y = groups) %>%
  group_by(g6_letter) %>%
  summarize(n = length(g6),
    Pomu = mean(Pomu),
    Syal.s = mean(Syal.s))
```

```
# A tibble: 6 × 4
  g6_letter      n    Pomu Syal.s
  <chr>      <int> <dbl> <dbl>
1 a          14 0.105  0.296
2 b           9 0.0689 0.773
3 c          10 0.594  0.294
4 d           3 0.0173 0.792
5 e           7 0.153  0.195
6 f           4 0      0.0135
```

For example, the mean abundance of Pomu in group 'a' is about a tenth of what it is in the stand where it was most abundant.

Once you understand how groups differ, you could incorporate that information into descriptive group names. These names could be used when group identities are overlaid onto ordinations, summarized in tables, etc. For example, Wainwright et al. (2019) conducted a hierarchical cluster analysis of vegetation data from permanent plots and identified four distinct groups of plots. They then examined characteristics of the species that were dominant in each group:

- Functional group (shrub, grass, forb)
- Nativity (whether native or non-native to area)
- Fire tolerance (whether shrubs were able to resprout or not after fire)

Based on this information, they named their four groups as obligate seeder, grass-forb, invaded sprouter, and pristine sprouter. See their Table 1 for more characteristics of these groups.

Overlaying Onto An Ordination

The multivariate set of responses used to create a distance matrix can also be visualized through an ordination. An ordination is a low-dimensional (often two-dimensional) representation of the dissimilarity matrix: points near one another are similar in composition, and points far from one another are more dissimilar. This graphical summary of the data can then be overlaid with other information, including about the groups generated through a cluster analysis.

Overlaying A Dendrogram (`vegan::ordicluster()`)

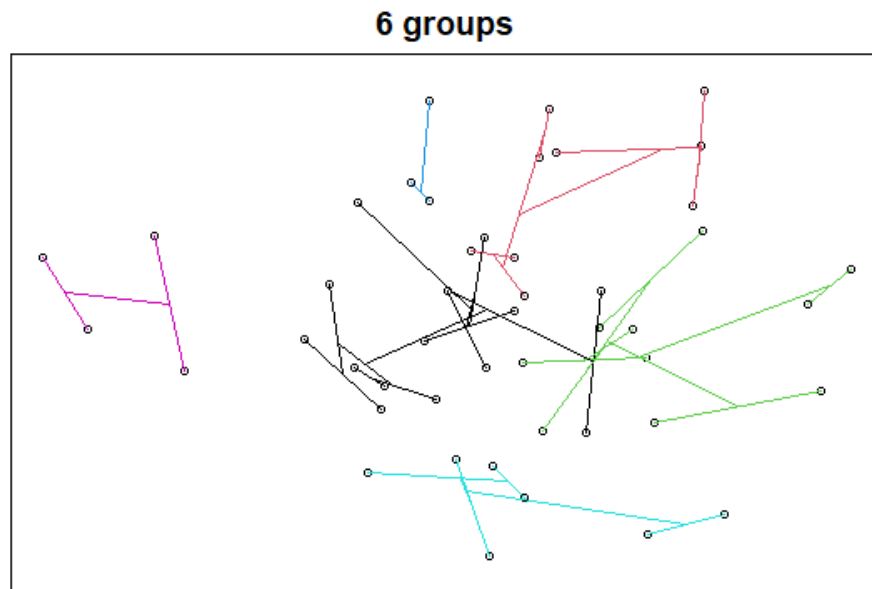
The `ordicluster()` function provides one way to explore the structure within the groups identified from a dendrogram. It simply adds lines to an ordination that connect plots in the order in which they were fused together in the cluster analysis. When a group is being fused, the line intersects the centroid of the group. Recall that a dendrogram can be envisioned as a (one-dimensional) mobile. What we're doing here is arranging the groups in two dimensions and then looking directly down onto that mobile.

We'll illustrate this using the oak plant community dataset, and the 6 groups that we identified from its dendrogram. We conduct and plot a NMDS ordination of the data, and then overlay the cluster results onto it:

```
Oak1.z <- metaMDS(comm = Oak1.dist,  
  k = 2)
```

```
plot(Oak1.z,  
  display = "sites",  
  main = "6 groups",  
  xaxt = "n",  
  yaxt = "n",  
  xlab = "",  
  ylab = "")
```

```
ordicluster(ord = Oak1.z,  
  cluster = Oak1.hclust,  
  prune = 5,  
  col = g6)
```



NMDS ordination of the oak plant community with the 6 groups identified in a hierarchical cluster analysis overlaid onto it. Colored lines connect stands within a group from the cluster analysis.

We'll cover NMDS ordinations soon. For now, please note that each point is a stand. Points that are near to one another are more similar in composition, and those that are far from one another are more different in composition.

A few notes about the `ordicluster()` function:

- The 'prune' argument specifies how many fusions to omit; since each fusion combines two groups, the number of groups to be displayed is one more than the number of fusions omitted (i.e., `prune = 5` shows 6 groups). In other words, the 'prune' argument omits the last or upper fusions in the dendrogram so that we can see the patterns within each group.
- For clarity, I used a different color for the lines connecting stands in each group.
- I used `g6` to index the colours; `groups$g6_1letter` did not work and I haven't debugged this.

This approach is not applicable to k-means cluster analysis ... do you see why?

Overlaying Group Perimeters (`vegan::ordihull()`)

Groups can also be highlighted in ways that do not require a dendrogram and that therefore can be applied to the groups identified through any type of cluster analysis. In fact, these methods can be applied to groups identified by any categorical variables!

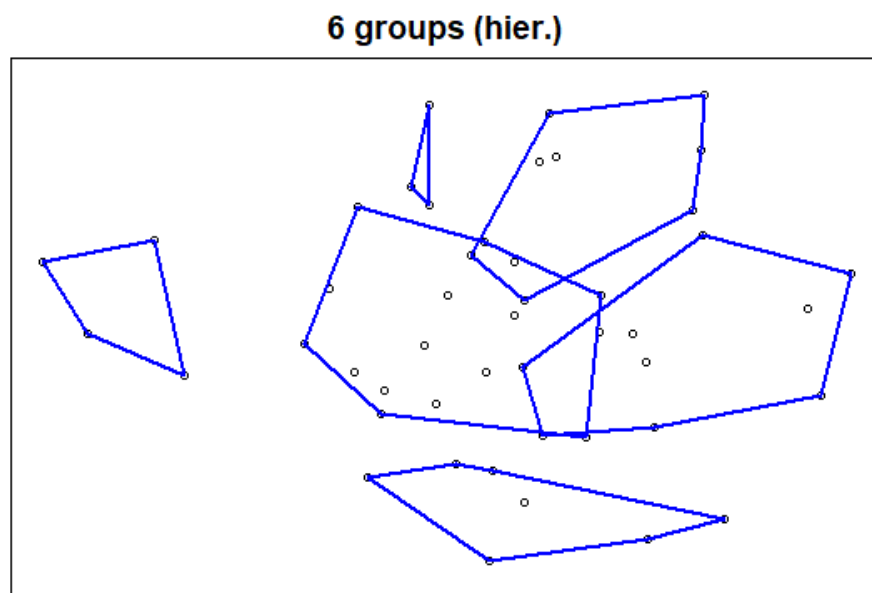
Here, we illustrate how to draw a polygon or hull that encompasses all sample units from the same group.

We plot the ordination again and then add hulls based on the grouping factor of interest. We can repeat this multiple times to explore different groupings.

First, let's view the six groups from our hierarchical cluster analysis:

```
plot(Oak1.z,  
     display = "sites",  
     main = "6 groups (hier.)",  
     xaxt = "n",  
     yaxt = "n",  
     xlab = "",  
     ylab = "")
```

```
ordihull(ord = Oak1.z,  
         groups = g6,  
         col = "blue",  
         lwd = 2)
```



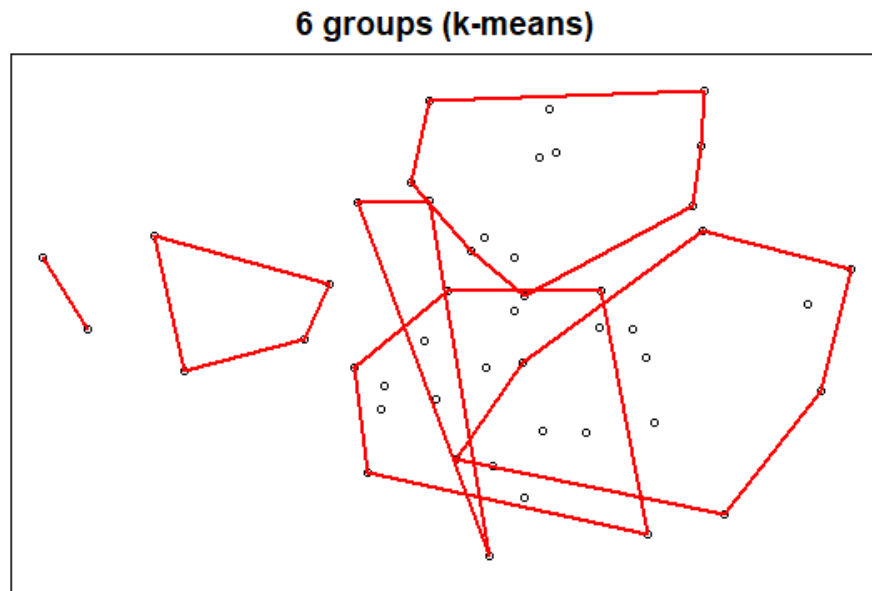
NMDS ordination of the oak plant community, with hulls delineating the 6 groups identified from a hierarchical cluster analysis of the stands. These groups are the same as was shown earlier by using `ordicluster()` to overlay the dendrogram onto the ordination.

Next, let's view the six groups from our *k*-means analysis:

```
plot(Oak1.z,  
     display = "sites",  
     main = "6 groups (k-means)",  
     xaxt = "n",  
     yaxt = "n",  
     xlab = "",  
     ylab = "")
```



```
ordihull(ord = Oak1.z,
  groups = k6,
  col = "red",
  lwd= 2)
```



NMDS ordination of the oak plant community, with hulls delineating the 6 groups identified from a k-means cluster analysis of the stands.

The two types of cluster analyses identified quite different groups, in part because one was hierarchical and the other was not.

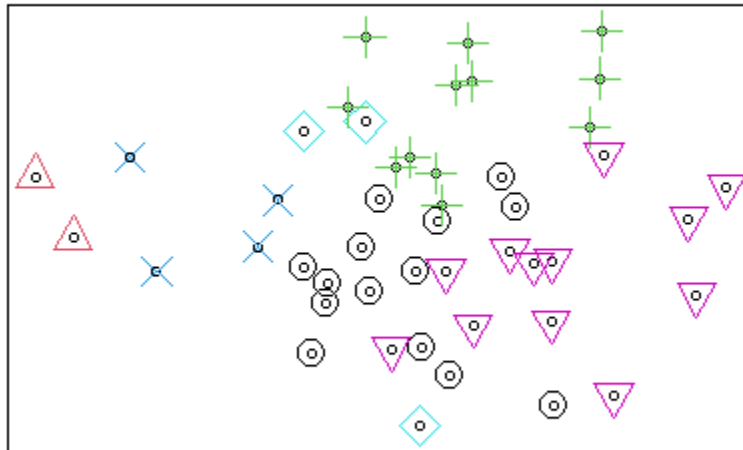
Overlaying Group Identities

We can use symbology to distinguish groups in an ordination. Let's use our k-means analysis to do so.

```
plot(Oak1.z,
  display = "sites",
  main = "6 groups (k-means)",
  xaxt = "n",
  yaxt = "n",
  xlab = "",
  ylab = "")

points(Oak1.z,
  pch = k6,
  col = k6,
  cex = 2)
```

6 groups (k-means)



NMDS ordination of the oak plant community, with symbol shapes and colors delineating the 6 groups identified from a k-means cluster analysis of the stands.

We will see more ways to visualize groups when we discuss ordinations.

Other Uses for Cluster Groups

The groups that we identify can serve as explanatory variables in subsequent analyses. For example:

- A cluster analysis of vegetation groups at one point in time might form a categorical variable in subsequent analyses of how vegetation communities changed over time. Mitchell et al. (2017) provide an example of this.
- Environmental variables might be tested to determine if they differ among groups defined from a cluster analysis of vegetation data, or vice versa.

Another option is to test for differences in the data matrix among groups. For example, if we are using a plant community data matrix as in our oak example dataset, we could test whether composition differs amongst the groups identified in a cluster analysis. This may be somewhat uninteresting, since the clustering was done expressly to identify groups that were as different from one another as possible. It is possible, I think, for these clusters to not differ statistically from one another, but in my experience this is uncommon. One other factor to keep in mind in this regard is the hierarchical structure of the groups: two groups from the same branch of the dendrogram are necessarily going to be more similar to each other than to a group from another branch of the dendrogram.

Although a cluster analysis identifies groups, that does not mean that all of the variables in the response matrix are equally different among those groups. Variables can be tested individually (or in subsets) to determine which group(s) they differ among. The ways that this is done need to reflect the characteristics of the data.

- Regular, continuously distributed variables can be analyzed using ANOVA or PERMANOVA.
- Community data are often represented by sparse matrices (lots of absences). One way to test for differences in this type of data is with Indicator Species Analysis (ISA). ISA considers both species presence/absence and species abundance, and identifies species that are more strongly associated with the groups than expected by chance. We'll consider this separately.

Conclusions

The groups identified through cluster analysis are based on the multivariate data collected on the sample units. Other information such as explanatory variables was not used in this classification.

The groups can be used to summarize variables collected on the same sample units, both variables that were included in the calculation of the distance matrix and those that were independent of it. This information can be used to assign more meaningful names to the groups.

Groups can be illustrated graphically by overlaying them onto an ordination based on the same data.

References

- Mitchell, R.M., J.D. Bakker, J.B. Vincent, and G.M. Davies. 2017. Relative importance of abiotic, biotic, and disturbance drivers of plant community structure in the sagebrush steppe. *Ecological Applications* 27:756-768.
- Wainwright, C.E., G.M. Davies, E. Dettweiler-Robinson, P.W. Dunwiddie, D. Wilderman, and J.D. Bakker. 2019. Methods for tracking sagebrush-steppe community trajectories and quantifying resilience in relation to disturbance and restoration. *Restoration Ecology* 28:115-126.

Media Attributions

- oak.tabasco
- ordicluster
- ordihull.g6
- ordihull.k6
- oak.kmeans

31. Discriminant Analysis

Learning Objectives

To assess how well observations have been classified into groups using linear methods (LDA) and distance-based methods (dbDA).

Key Packages

```
require(MASS, WeDiBaDis)
```

Introduction

Techniques such as cluster analysis are used to identify groups *a posteriori* based on a suite of correlated variables (i.e., multivariate data). This type of analysis is sometimes followed by an assessment of how well observations were classified into the identified groups, and how many were misclassified. This assessment is known as a discriminant analysis (DA) (aka canonical analysis or supervised classification). Williams (1983) notes that DA can be conducted for two reasons:

- **Descriptive** – to separate groups optimally on the basis of the measurement values
- **Predictive** – to predict the group to which an observation belongs, based on its measurement values

In these notes, I demonstrate linear and distance-based DA techniques.

- **Linear discriminant analysis** (LDA) is the most common method of DA. It is an eigenanalysis-based technique and therefore is appropriate for normally-distributed data.
- As implied by the name, **distance-based discriminant analysis** (Anderson & Robinson 2003) can be applied to any type of data, including community-level data and data types such as categorical variables that cannot be incorporated into LDA. Irigoien et al. (2016) describe how these techniques can be used in biology (e.g., morphometric analysis for species identification), ecology (e.g., niche separation by sympatric species), and medicine (e.g., diagnosis of diseases).

Linear Discriminant Analysis (LDA)

Theory

Rather than relating individual variables to group identity, LDA identifies the combination of variables that best discriminates (distinguishes) sample units from different groups. Combinations of variables can be more sensitive than individual variables. In the below figure (Figure 11.11 from Legendre & Legendre 2012), analyses that focus just on **x1** or just on **x2** cannot distinguish these groups (compare the histograms shown along the **x1** and **x2** axes). However, when **x1** and **x2** are combined into a new composite variable **z**, that variable clearly distinguishes the two groups (see histogram along the **z** axis).

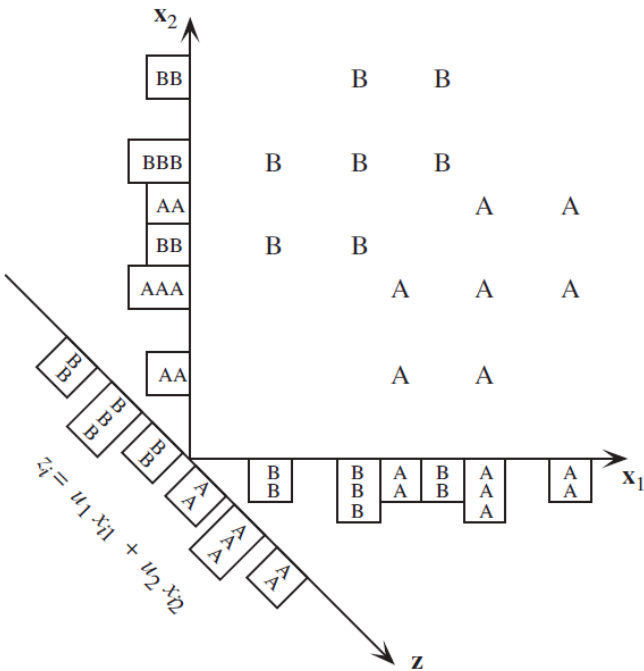


Figure 11.11 Two groups, A and B, with 6 objects each, are overlapping on both axes, x_1 and x_2 , as shown by the histograms on the axes. They are perfectly separated, however, along a discriminant axis z . The position of each object i is calculated along z using the equation $z_i = (\cos 45^\circ) x_{i1} - (\cos 45^\circ) x_{i2}$. Adapted from Jolicoeur (1959).

LDA is an eigenanalysis technique (refer back to the chapter about matrix algebra if that term doesn't sound familiar; we'll discuss eigenanalysis in more detail when we focus on PCA) that also incorporates information about group identity. Conceptually, it can be thought of as a MANOVA in reverse:

	Explanatory (Independent) Variable(s)	Response (Dependent) Variable(s)
MANOVA	Group Identity	Data
LDA	Data	Group Identity

Given the similarity between LDA and MANOVA, it is perhaps unsurprising that LDA has the same assumptions with respect to multivariate normality, etc. It can be applied to data that satisfy these assumptions; environmental and similar types of data might meet these assumptions, but community-level data do not. It is important to keep in mind that it identifies **linear** combinations of variables.

For more information on LDA, see chapter 8 in Manly & Navarro Alberto (2017) and section 11.3 in Legendre & Legendre (2012).

Why Use LDA?

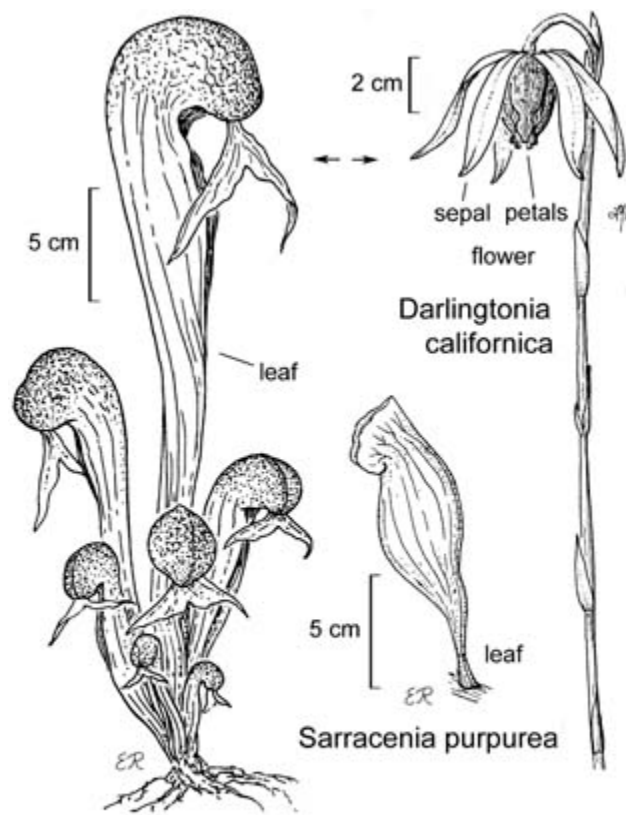
LDA has been used for various ecological applications, as described by McCune & Grace (2002, p. 205). Williams (1983) provides citations of some older ecological examples of LDA. Leva et al. (2009) provide an ecological example of how LDA might be applied. In their study, they measured a number of traits of the roots of eight grasses and then used LDA to classify observations to species on the basis of these traits. They also used cluster analysis and PCA to help understand the correlations between these traits and relationships among the species.

The most common reason to use LDA is to assess how well observations from different groups can be differentiated from one another on the basis of their measured responses. For example, the below dataset involves a bunch of measurements on a number of *Darlingtonia* plants. How well do these variables help us identify the site at which they were collected?

Here's another potential application: suppose a field assistant measured another *Darlingtonia* plant but forgot to record the site they were working at. Their datasheet contains all of the measured variables except the site ID, so this plant isn't included in our dataset. Can we use the relationships among the measured variables to predict which site the plant came from?

Example Dataset: *Darlingtonia californica*

We'll illustrate LDA using detailed morphological measurements from 87 *Darlingtonia californica* (cobra lily; California pitcherplant) plants at four sites in southern Oregon. These data are from Table 12.1 of Gotelli & Ellison (2004), and are the same ones that we use to illustrate Principal Component Analysis (PCA).



Darlingtonia californica

Sarraceniaceae

© Regents of the University of California

Line drawing of *Darlingtonia californica* (California pitcherplant).

In this plant, leaves are modified into tubes that can hold liquid and trap insects. The 10 measured variables (and their units) are:

Name	Type	Description	Units
height	Morphological	Plant height	mm
mouth.diam	Morphological	Diameter of the opening (mouth) of the tube	mm
tube.diam	Morphological	Diameter of the tube	mm
keel.diam	Morphological	Diameter of the keel	mm
wing1.length	Morphological	Length of one of the two wings in the fishtail appendage (in front of the opening to the tube)	mm
wing2.length	Morphological	Length of second of the two wings in the fishtail appendage (in front of the opening to the tube)	mm
wingsprea	Morphological	Length from tip of one wing to tip of second wing	mm
hoodmass.g	Biomass	Dry weight of hood	g
tubemass.g	Biomass	Dry weight of tube	g
wingmass.g	Biomass	Dry weight of fishtail appendage	g

The datafile is available through the GitHub repository. We begin by importing the data.

```
darl <- read.csv("data/Darlingtonia_GE_Table12.1.csv")
```

The variables differ considerably in terms of the units in which they are measured, so it makes sense to normalize them (i.e., convert them to Z-scores) before analysis:

```
darl.data <- scale(darl[,3:ncol(darl)])
```

However, this is much less important for LDA than it is for PCA. To verify that the results of an LDA are not affected by this normalization, try running the below analyses using unscaled (raw) data from these *Darlingtonia* plants.

LDA in R (`MASS::lda()`)

We will use the `lda()` function in the `MASS` package, though other options also exist. The usage of `lda()` is:

```
lda(x,
    grouping,
    prior = proportions,
    tol = 1.0e-4,
    method,
    CV = FALSE,
    nu,
    ...
)
```

The key arguments are:

- `x` – the matrix or data frame containing the explanatory variables. This and the `grouping` argument could be replaced by a formula of the form `groups ~ x1 + x2 + ...`. Note that this formula is the opposite of that for an ANOVA – the grouping variable is the response here.
- `grouping` – a factor specifying the group that each observation belongs to.
- `prior` – prior probabilities of group membership. Default is to set probabilities equal to sample size (i.e., a group that contains more observations will have a higher prior probability).
- `tol` – tolerance to decide whether the matrix is singular (based on whether variance is smaller than `tol^2`). Default is 0.0001.
- `method` – how to calculate means and variances. Options:
 - `moment` – standard approach (means and variances), and the default.
 - `mle` – via maximum likelihood estimation
 - `mve` – for a subset of the data that you specify (to exclude potential outliers)
 - `t` – based on a *t*-distribution.
- `CV` – whether to perform cross-validation or not. Default is to not (`FALSE`). Note that this argument is spelled with capital letters.
- `nu` – degrees of freedom (required if `method = "t"`).

The resulting object includes several key components:

- **prior** – The prior probabilities of class membership used. As noted above, the default is that these are proportional to the number of sample units in each group. See ‘Misclassifications’ section below for more details about this.
- **means** – The mean value of each variable in each group (be sure to note whether these are standardized or unstandardized).
- **scaling** – A matrix of coefficients (eigenvectors) of **linear discriminant (LD) functions**. Note that there is one fewer LD than there were groups in the dataset (because if you know that an observation doesn’t belong to one of these groups then it must belong to the last group). These are roughly analogous to the loadings produced in a PCA. Since we standardized our data before analysis, we can compare the LD coefficients directly among variables. Larger coefficients (either positive or negative) indicate variables that carry more weight with respect to that LD. The position of a given observation on a LD is calculated by matrix multiplying the measured values for each variable by the corresponding coefficients.
- The proportion of variation explained by each LD function (eigenvalue). Note that these always sum to 1, and are always in descending order (i.e., the first always explains the most variation).

Applying `lda()` to the *Darlingtonia* Data

Let’s load the MASS package and then carry out an LDA of the *Darlingtonia* data:

```
library(MASS)
darl.da <- lda(x = darl.data, grouping = darl$site)
darl.da <- lda(darl$site ~ darl.data) # Equivalent
darl.da
```

```
Call:
lda(darl$site ~ darl.data)

Prior probabilities of groups:
      DG      HD      LEH      TJH
0.2873563 0.1379310 0.2873563 0.2873563

Group means:
      darl.dataheight darl.datamouth.diam darl.datatube.diam darl.datakeel.diam
DG          0.03228317          0.34960765          -0.64245369          -0.3566764
HD         -0.41879232         -1.37334175           0.93634832           1.3476695
LEH          0.22432324          0.25108333           0.23567347          -0.4189580
TJH         -0.05558610          0.05851306          -0.04266698           0.1287530
      darl.datawing1.length darl.datawing2.length darl.datawingsprea
DG          0.3998415          0.2997874          -0.2093847
HD         -0.8102232          -0.3184490          -0.1996899
LEH          0.5053448          0.3853959           0.6919511
TJH         -0.5162792          -0.5323277          -0.3867152
      darl.datahoodmass.g darl.datatubemass.g darl.datawingmass.g
DG          0.4424329          0.30195655           0.03566704
HD         -1.1498678          -1.05297022          -0.28283531
LEH         -0.1754338          0.07558687           0.23585050
TJH          0.2849374          0.12788229          -0.13575660

Coefficients of linear discriminants:
      LD1      LD2      LD3
darl.dataheight      1.40966787  0.2250927 -0.03191844
darl.datamouth.diam  -0.76395010  0.6050286  0.45844178
darl.datatube.diam    0.82241013  0.1477133  0.43550979
```

```

darl.datakeel.diam      -0.17750124 -0.7506384 -0.35928102
darl.dataawing1.length  0.34256319  1.3641048 -0.62743017
darl.dataawing2.length -0.05359159 -0.5310177 -1.25761674
darl.dataawingsprea     0.38527171  0.2508244  1.06471559
darl.datahoodmass.g     -0.20249906 -1.4065062  0.40370294
darl.datatubemass.g     -1.58283705  0.1424601 -0.06520404
darl.dataawingmass.g     0.01278684  0.0834041  0.25153893

Proportion of trace:
      LD1      LD2      LD3
0.5264 0.3790 0.0946

```

You should be able to link parts of this output to the key components produced by `lda()` in its description above. Each piece can also be indexed or calculated separately, of course.

The prior probabilities here are simply the number of samples from a site divided by the total number of samples. They therefore have to sum to 1.

The coefficients of linear discriminants – an eigenvector – would be multiplied by the corresponding variables to produce a score an individual observation on a particular LD. To view just the matrix of coefficients for each LD function:

```
darl.da$scaling %>% round(3)
```

```

               LD1      LD2      LD3
darl.dataheight    1.410  0.225 -0.032
darl.datamouth.diam -0.764  0.605  0.458
darl.datatube.diam  0.822  0.148  0.436
darl.datakeel.diam -0.178 -0.751 -0.359
darl.dataawing1.length 0.343  1.364 -0.627
darl.dataawing2.length -0.054 -0.531 -1.258
darl.dataawingsprea  0.385  0.251  1.065
darl.datahoodmass.g -0.202 -1.407  0.404
darl.datatubemass.g -1.583  0.142 -0.065
darl.dataawingmass.g  0.013  0.083  0.252

```

The larger these values are (in absolute terms), the more important that variable is for determining the location of a plant along that LD. For example, plant height is much more important for LD1 than for LD2 or LD3.

The values labeled as ‘proportion of trace’ on the screen are calculated as:

```
(darl.da$svd^2/sum(darl.da$svd^2)) %>% round(3)
```

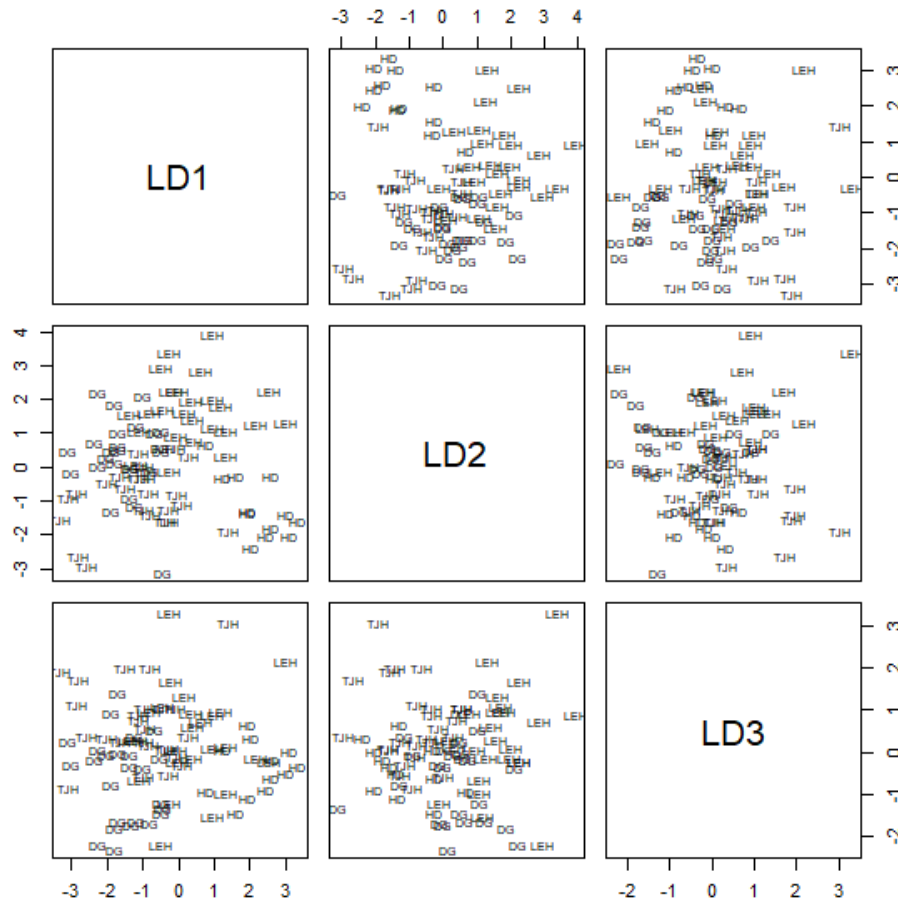
```
0.526 0.379 0.095
```

These proportions – the eigenvalues – are associated with the LDs, and are always in descending order (LD1 > LD2 > LD3 > ...). They also sum to 1. The first LD is the important important for discriminating between the groups, and LD3 is least important of these three.

Plotting a LDA

To graph all pairwise combinations of LDs:

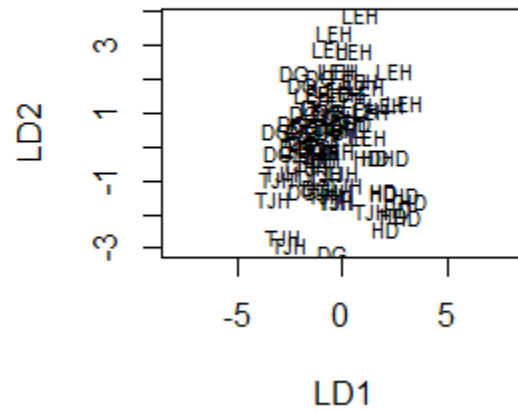
```
plot(darl.da) #pairs(darl.da) is equivalent
```



Scatterplot of pairs of LDs. Each point is a Darlingtonia plant, abbreviated by the code of the site from which it was collected.

We can add the 'dimen' argument to specify how many LDs to plot. To plot LD1 vs LD2:

```
plot(darl.da, dimen = 2)
```

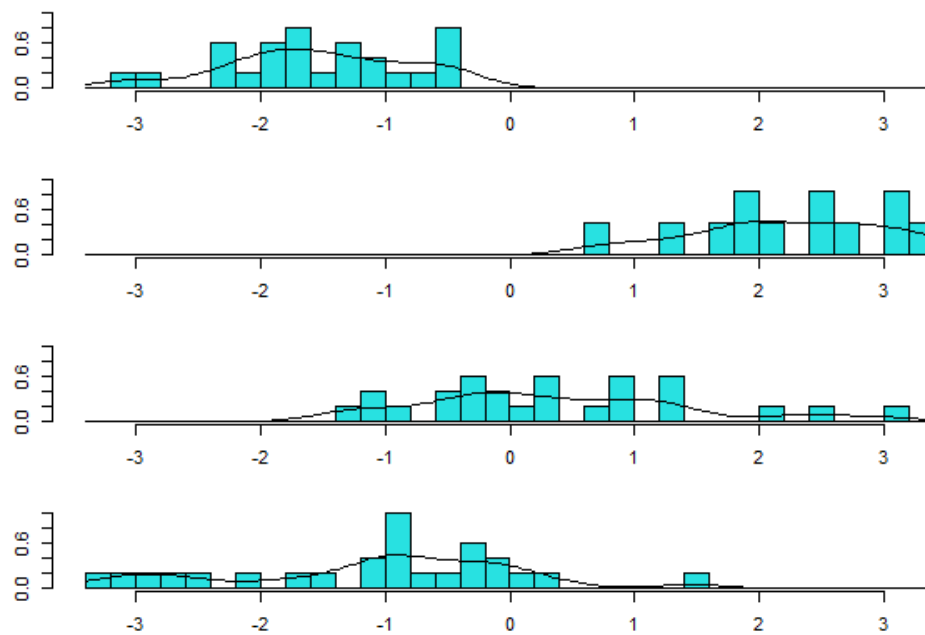


Scatterplot of LD1 vs LD2. Each point is a *Darlingtonia* plant, abbreviated by the code of the site from which it was collected.

To plot just LD1 for each group:

```
plot(darl.da, dimen = 1, type = "both")
```

The 'type' argument here specifies whether to graph "histogram", "density", or "both".



LD1 values for the *Darlingtonia* plants, by site.

Misclassifications

How well were we able to predict which site an individual came from on the basis of these morphological data? We can use the `predict()` function to predict the group that each individual belongs to:

```
darl.predict <- predict(darl.da)
```

This function returns three components:

- `class` – The predicted group identity of each observation.
- `posterior` – The posterior probability that each observation belongs to each of the four sites. Note that larger values are better, and the plant is predicted to belong to the site in which it has the largest posterior value. For any observation, the posterior probabilities of the different groups will sum to 1.
- `x` – The scores of each observation for each LD.

For example, here are the predicted group identities of the first few observations:

```
head(darl.predict$class)
```

```
[1] DG  TJH LEH TJH DG  TJH
Levels: DG HD LEH TJH
```

The first observation was predicted to belong to site DG, the second to site TJH, etc.

How well were observations classified according to site? We can summarize this in a cross-classification table (aka **confusion matrix**), the results of which are shown below:

```
darl.table <- table(darl$site, darl.predict$class)
darl.table
```

	DG	HD	LEH	TJH
DG	18	0	2	5
HD	0	11	1	0
LEH	3	0	21	1
TJH	2	0	3	20

The rows in this table correspond to the true classification, and the columns to the predicted classification from the DA. The row totals (not shown) are the actual numbers of observations in each group. Values along the diagonal are correctly predicted; all others are misclassified. The percentage of observations that were correctly classified is:

```
sum(diag(darl.table))/sum(darl.table) * 100
```

So, about 80% of observations were correctly classified. You could also index this calculation to figure out this percentage separately for each site.

Prior Probabilities

One important aspect of a DA is the decision of what **prior probabilities** to use. These probabilities are weights in the calculations of a DA: the higher the probability, the more weight a group is given. We often assume that the prior probabilities are either i) equal among groups, or ii) proportional to sample sizes. Unless otherwise specified, the `lda()` function assumes that prior probabilities are equal to sample sizes.

The prior probabilities can alter the likelihood and direction of **misclassification**. Whether the direction matters depends on the situation. McCune & Grace (2002, p. 209-210) give an example in which the prior probabilities altered the classification of sites as goshawk nesting habitat or not. These differences could easily affect land management. More significant consequences can occur in medical studies: for example, consider the difference between misclassifications that predict someone has cancer when they don't, and misclassifications that predict they don't have cancer when they actually do. Which would you prefer?

Cross-Validation

The above confusion matrix permitted us to assess the results of a DA by comparing them against the observed data. Data can also be **cross-validated** in various ways:

- **Jackknifing.** In `lda()`, this can be done by adding the '`CV = TRUE`' argument to the function. Jackknifing involves the following steps:
 - omit one observation from dataset
 - analyze reduced dataset
 - use results to classify the omitted observation
 - repeat for next observation in dataset
- Reserving a subset of the data as an independent validation dataset. These data are not used in the initial analysis, but are used to assess how well the classification works. This is only feasible if the dataset is large.
- Collect new data and use DA to classify it (though there needs to be some way to independently verify that they are classified correctly).

Jackknifing the darl data:

```
darl.da.CV <- lda(darl$site ~ darl.data, CV = TRUE)
```

We can use a confusion matrix to compare the results of the cross-validated LDA with the original classification (or with the results of the LDA without cross-validation).

```
table(darl$site, darl.da.CV$class)
```

	DG	HD	LEH	TJH
DG	17	1	2	5
HD	0	10	2	0
LEH	4	1	18	2
TJH	3	1	3	18

With cross-validations, the accuracy has declined to 72% (ensure you understand how this was calculated).

We can also compare the posterior probabilities produced via jackknifing with those predicted from the original LDA based on all data. For the first observation (and rounding probabilities to 3 decimal places):

```
darl.da.CV$posterior[1,] %>% round(3)
```

	DG	HD	LEH	TJH
	0.875	0.000	0.000	0.124

```
darl.predict$posterior[1,] %>% round(3)
```

	DG	HD	LEH	TJH
	0.772	0.000	0.000	0.228

To compare all observations, we might focus on the differences between the probabilities:

```
(darl.da.CV$posterior - darl.predict$posterior) %>%  
  round(3) %>%  
  head()
```

	DG	HD	LEH	TJH
1	0.104	0.000	0.000	-0.104
2	0.086	0.000	0.021	-0.107
3	0.002	0.004	0.060	-0.066
4	0.052	0.000	0.002	-0.054
5	0.074	0.000	0.009	-0.084
6	0.057	0.000	0.001	-0.058

Note that a reduced posterior probability for one or more groups must be accompanied by an increased posterior probability in one or more other groups. Large differences indicate observations that strongly affect the LDA; the result of the jackknifing differs considerably when they are omitted.

Key Takeaways

Cross-validation is important for assessing how well the products of a LDA can classify independent observations.

Distance-based Discriminant Analysis

(WeDiBaDis::WDBdisc())

Anderson & Robinson (2003) present an alternative method for conducting a DA on the basis of a distance matrix. Since any distance measure can be used to calculate the distance matrix, they suggest this method can be used for community-level data. This appears to involve using metric multidimensional scaling to reduce the dimensionality of the data and express it in Euclidean units, followed by a LDA on the locations of the sample units in the reduced ordination space.

Irigoiien et al. (2016) provide a package, WeDiBaDis (Weighted Distance Based Discriminant Analysis) that provides this functionality. It is available through their GitHub page:

```
library(devtools)
install_github("ItziarI/WeDiBaDis")
```

Load the package once it's installed:

```
library(WeDiBaDis)
```

The usage of the key function, `WDBdisc()`, is as follows:

```
WDBdisc(data,
  datatype,
  classcol = 1,
  new.ind = NULL,
  distance = "euclidean",
  method,
  type
)
```

The key arguments are:

- `data` – a data matrix or distance matrix, with a column containing the class labels (grouping factor). By default, the class labels are assigned to be in the first column; this is set by the `classcol` argument.
- `datatype` – 'm' if data are a data matrix, 'd' if data are a distance matrix.
- `classcol` – column number in which actual classification is coded. Default is the first column.
- `new.ind` – whether or not there are new individuals to be classified.
- `distance` – type of distance measure to be used if data is a data matrix. Options are `euclidean` (default), `correlation`, `Bhattacharyya`, `Gower`, `Mahalanobis`, `BrayCurtis`, `Orloci`, `Hellinger`, and `Prevosti`.
- `method` – whether to conduct distance-based DA (DB) or weighted distance-based DA (WDB). I haven't delved into the basis of the weighting, but it can be helpful if the sample sizes differ among groups.

The output includes a number of elements:

- `conf` – the confusion matrix relating the actual classification to the classification as predicted by the explanatory variables. This is the same type of table that we calculated above.
- `pred` – predicted class for observations that did not have an actual classification, if any were included.

A visual representation of the confusion matrix can be seen via the `plot()` function.

The `summary()` function can be applied to the output and produces additional information:

- Total correct classification: the sum of the diagonals of the confusion matrix divided by the total number of observations (as calculated above).
- Generalized squared correlation: ranges from 0 to 1. A value of 0 means that there is at least one class in which no units are classified.
- Cohen's Kappa coefficient: a measure of the agreement of classification to the true class. Irigoien et al. (2016) provide the following rules of thumb for interpreting it:
 - < 0: less than chance agreement
 - 0.01-0.20: slight agreement
 - 0.21-0.40: fair agreement
 - 0.41-0.60: moderate agreement
 - 0.61-0.80: substantial agreement
 - 0.81-0.99: almost perfect agreement
- Sensitivity (or recall), for each class. The number of observations correctly predicted to belong to a given class, as a percentage of the total number of observations in that class.
- Precision (positive predictive value), for each class. Probability that a classification in this class is correct.
- Specificity, for each class. A measure of the ability to correctly exclude an observation from this class when it really belongs to another class.
- F1-score, for each class. Calculated based on sensitivity and precision.

The sensitivity, precision, and specificity can also be graphed by plotting the output of this summary function.

Applying `WDBdisc()` to the *Darlingtonia* data

Note that this function requires the data in a specific format that contains both the response (classification) and the explanatory variables. In this case, we'll provide the explanatory variables and allow the function to apply the Euclidean distance matrix to it (with other datasets, we could specify a different distance measure or provide a matrix of distances). And, since the number of observations differs among sites, we'll use the weighted DA.

```
darl.WDB <- cbind(darl$site, darl.data)
```

```
darl.dbDA <- WDBdisc(data = darl.WDB,  
  datatype = "m",  
  classcol = 1,  
  method = "WDB")
```

```
summary(darl.dbDA)
```

```
Discriminant method:  
----- Leave-one-out confusion matrix: -----  
      Predicted  
Real   DG  HD  LEH  TJH  
  DG   11   0   8   6  
  HD    0  10   0   2  
  LEH   0   3  20   2  
  TJH   3   6   6  10
```

```

Total correct classification:  58.62 %

Generalized squared correlation:  0.2632

Cohen's Kappa coefficient:  0.4447793

Sensitivity for each class:
      DG      HD      LEH      TJH
44.00 83.33 80.00 40.00

Predictive value for each class:
      DG      HD      LEH      TJH
78.57 52.63 58.82 50.00

Specificity for each class:
      DG      HD      LEH      TJH
64.52 54.67 50.00 66.13

F1-score for each class:
      DG      HD      LEH      TJH
56.41 64.52 67.80 44.44
-----

No predicted individuals

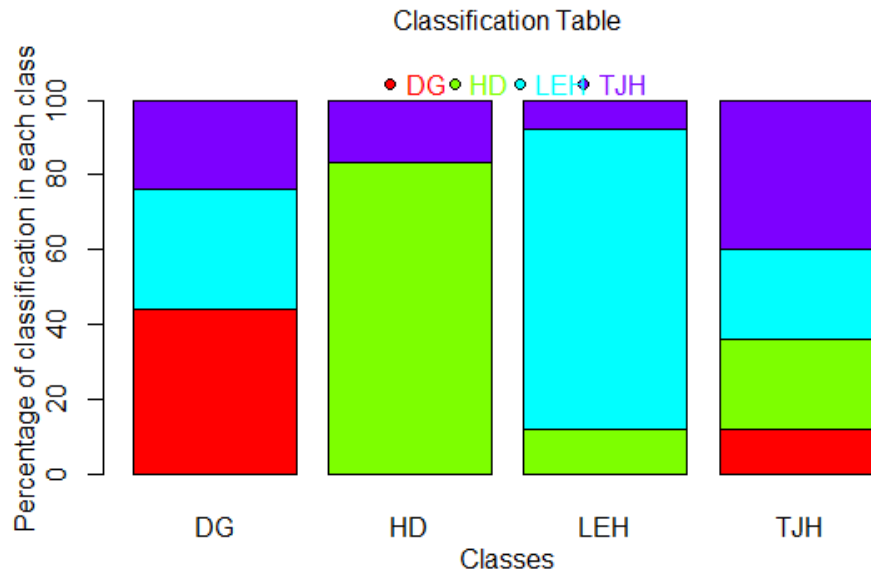
```

Note that the confusion matrix is a leave-one-out confusion matrix, which means that jackknifing has automatically been performed.

Just under 60% of the observations are being correctly classified. Cohen's Kappa indicates moderate agreement between the actual classification and the classifications based on these variables. Observations at the HD and LH sites are particularly well classified.

To see a visualization of the confusion matrix:

```
plot(darl.dbDA)
```

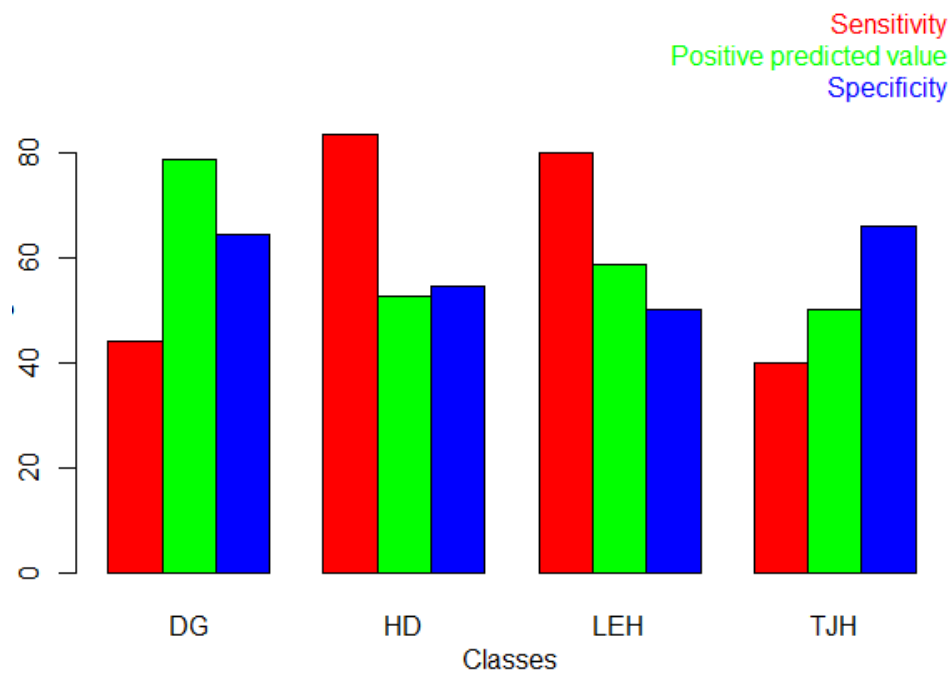


Classification table arising from distance-based Discriminant Analysis of 87 Darlingtonia plants in four sites.

The bars in this plot are the actual sites (labeled 'Classes'). The colors also distinguish sites. The portion of a bar that is the same color as the label beneath the bar is the plants that were correctly classified to the site that they actually came from. Other colors within a bar indicate the other sites to which plants from that site were classified. For example, most of the plants from site LEH (light blue color) were correctly classified, and none of the plants from this site were classified as belonging to site HD.

Finally, the `summary()` output can also be plotted:

```
plot(summary(darl.dbDA))
```



*Classification table arising from distance-based Discriminant Analysis of 87 *Darlingtonia* plants in four sites.*

This summarizes the sensitivity, positive predicted value, and specificity for each group.

Key Takeaways

Distance-based discriminant analysis provides a method for classifying observations in situations where assumptions about linearity are not appropriate.

Conclusions

Discriminant analysis is an important tool for understanding differences among groups and using those differences to predict (classify) the group to which other samples belong. It is common for example in medical statistics in terms of evaluating the accuracy of diagnoses.

The medical context is also helpful for recognizing that there may be dramatically different consequences for different types of misclassification – for example, the distinction between classifying someone as having a disease when they actually don't (false positive) and classifying someone as not having a disease when they actually do (false negative) (Irigoien et al. 2016).

References

- Anderson, M.J., and J. Robinson. 2003. Generalized discriminant analysis based on distances. *Australian and New Zealand Journal of Statistics* 45:301-318.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.
- Irigoien, I., F. Mestres, and C. Arenas. 2016. Weighted distance based discriminant analysis: the R package WeDiBaDis. *The R Journal* 8:434-450.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.
- Leva, P.E., M.R. Aguiar, and M. Oesterheld. 2009. Underground ecology in a Patagonian steppe: root traits permit identification of graminoid species and classification into functional types. *Journal of Arid Environments* 73:428-434.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- McKenna Jr., J.E. 2003. An enhanced cluster analysis program with bootstrap significance testing for ecological community analysis. *Environmental Modelling & Software* 18:205-220.
- Williams, B.K. 1983. Some observations on the use of discriminant analysis in ecology. *Ecology* 64:1283-1291.

Media Attributions

- Legendre.Legendre.2012_Figure.11.11
- Darlingtonia.californica_Jepson
- LDA.pairs
- LDA.2D
- LDA.1D
- dbDA.classification.table
- dbDA.summary

32. Overview of Classification and Regression Trees

Learning Objectives

- To consider ways of classifying observations based on both response and explanatory variables.
- To understand the difference between classification trees and regression trees.

Key Packages

```
require(tidyverse, mvpart)
```

Introduction

Classification and regression trees have the same objective as cluster analysis – to classify observations into groups on the basis of responses – but differ from cluster analysis in that explanatory variables are also incorporated into the classification. As a result, classification and regression trees are also known as **constrained clustering** or **supervised clustering** (Borcard et al. 2018). Borcard et al. (2018) also note that classification and regression trees have a strong focus on prediction, whereas unconstrained ordination methods focus on explanatory ability.

Classification and regression trees also differ from cluster analysis because they are divisive (vs. cluster analysis, which is agglomerative). In other words, they start with all sample units in one group and search for the best ways to split the group. This concept is why classification and regression trees are also known as **recursive partitioning**.

The basic idea of a hierarchical, tree-based model is familiar to most ecologists – a dichotomous taxonomic key is a simple example of one. Classification and regression trees are techniques that have entered the ecological literature relatively recently. Computationally, they can be thought of as amalgams of multiple regression, cluster analysis, discriminant analysis, and other techniques.

Classification trees refer to analyses that use categorical data for the response variable, while **regression trees** refer to analyses that use continuous data for the response variable. Although these types of data are distinguished in the terminology, please note that the same function can be used to analyze both, recognizing them by the class of the data. For simplicity, I will refer to them simply as regression trees in these notes. *Note:* These techniques are sometimes known as CART

(Classification and Regression Trees), which is the proprietary name of a regression tree program (<https://www.minitab.com/en-us/products/spm/>).

De'ath (2002) notes that regression trees can be used to explore and **describe** the relationships between species and environmental data, and to **classify** (predict the group identity of) new observations. More generally, regression trees seek to relate response variables to explanatory variables by locating groups of sample units with similar responses in a space defined by the explanatory variables. Unique aspects of regression trees are that the ecological space can be nonlinear and that they can easily include interactions between environmental variables. The environmental variables can also be of any type (categorical, ordinal, continuous); they will be treated appropriately based on their class.

The original key citation about regression trees is Breiman et al (1984). Helpful references about regression trees are De'ath (2002), De'ath & Fabricius (2000), Vayssières et al (2000), Venables & Ripley (2002, ch. 9), and Everitt & Hothorn (2006, ch. 8).

McCune & Grace (2002, ch. 29) give a few ecological applications of regression trees. Vayssières et al (2000) used univariate regression trees to predict the distributions of three major *Quercus* species in California. Chase & Rothley (2007) used regression trees to predict, on the basis of 10 biogeoclimatic and positional variables, sites where grasslands and heathlands do not presently occur but could be established. Smith et al. (2019) use multivariate regression trees to model water resource systems and compare the tradeoffs associated with planning decisions.

Regression trees can be conducted with both univariate and multivariate data (De'ath 2002). We will use univariate regression trees to explore the basic concepts and then extend those concepts to multivariate regression trees.

Key Takeaways

Classification and regression trees can **describe** the relationship between existing variables and to **predict** the group identity of new observations. They do so through a **divisive** process, identifying the value of an explanatory variable that best separates a group of responses into two sub-groups or branches. This process is **hierarchical**: early splits can have cascading consequences throughout the rest of the tree.

Spider Data (and Installing `mvpart`)

For this example, we'll begin by analyzing the relationship between the abundance of a hunting spider, *Trochosa terricola*, and six environmental variables. The data file is included in the `mvpart` package.



© Stefan Soltoffs, www.eurospiders.com

Trochosa terricola, a hunting spider.

The `mvpart` package is not actively maintained and therefore is no longer available through the standard installation routines. Instead, we have to install it from a GitHub archive:

```
install.packages("devtools")  
  
devtools::install_github("cran/mvpart")
```

Compiling this package may require that `Rtools` also be installed. This install should happen automatically but takes a few minutes, and afterwards you may have to re-run the command to install `mvpart`. If it doesn't happen automatically, you may have to load it manually from outside of R.

Once the package has been compiled and installed, it can be loaded. We'll load the `tidyverse` at the same time.

```
library(mvpart, tidyverse)
```

Now, we can load the spider data using the `data()` function:

```
data(spider)
```

(Note: if you are unable to load `mvpart`, you can still obtain the spider data from the 'data' folder in the GitHub repository. In that case, you can load it using `spider <- read.csv("data/spider.csv", row.names = 1)`)

```
dim(spider)
```

```
[1] 28 18
```

The data frame contains 18 columns: 12 containing the abundances of various spider species and 6 containing environmental variables (`water`, `sand`, `moss`, `reft` [light], `twigs`, `herbs`). Each variable is quantified on an ordinal scale from 0 to 9.

We're going to call the explanatory variables together, so let's create a new object containing just them:

```
variables <- c("water", "sand", "moss", "reft", "twigs", "herbs")  
  
env <- spider %>% select(any_of(variables))
```

We'll use these data to illustrate univariate regression trees and then extend this to multivariate regression trees.

Advantages and Disadvantages of Regression Trees

Advantages of regression trees include:

- Robust: minimal model assumptions:
 - Do not need to assume normality
 - Species-environment relationships can be nonlinear
 - Don't need to make simplifying assumptions about the data. For example, parametric models assume there is a single dominant structure in the data, whereas regression trees work with data that might have multiple structures.
- Can include continuous and discrete explanatory variables in the same model
- Can identify interactions among explanatory variables.
- No need to preselect variables to include in model; uses automatic stepwise variable selection.
- Variables can be reused in different parts of the tree. At each stage, the variable selected is the one "holding the most information for the part of the multivariate space it is currently working on" (Vayssières et al 2000, p. 683).
- Relatively insensitive to outliers.
- Unaffected by monotonic transformations of explanatory variables (only rank order matters for splitting)
- Easily interpretable outputs.

Disadvantages of regression trees include:

- Represent continuous variables by breaking them into subsets and assuming the same average value for all observations within a subset. In other words, an intercept-only model is fit to each subset. Therefore, regression trees may mask linear relationships within the data.
- Later splits are based on fewer cases than the initial ones.
- Splits at the top of the tree are more important than those that occur near the leaves.
- Generally require large sample sizes, depending on how many observations are required in each leaf (see `minsplit` and `minbucket` arguments below).
- Ability to conduct cross-validation is a function of sample size.

Boosted Trees, Forests, and Beyond

These notes provide a very simple introduction to the idea of classification and regression trees. Many other R packages are available for regression trees, including:

- `VSURF` – to help identify which variables to focus on (Genuer et al. 2015).
- `treeClust` – proposes a way to use classification and regression trees to calculate dissimilarities among objects that can then be used to cluster the objects using PAM (Buttrey & Whitaker 2015).

- **partykit** – non-parametric regression trees through the function `ctree()` (Hothorn et al. 2006).
- **IntegratedMRF** – univariate and multivariate random forests.

Many variations and extensions are available, including **boosted regression trees** (De'ath 2007, Elith et al. 2008; Parisien & Moritz 2009), **random forests** (Cutler et al. 2007; Van Kane will discuss these), and **cascade multivariate regression trees** (Ouellette et al. 2012). Essentially, these techniques involve conducting multiple regression trees and then averaging the results to yield a final 'ensemble' solution. These are computationally intensive procedures, but appear to produce models that have stronger predictive capabilities than single regression trees. Because they require the construction of large numbers of trees, they require large datasets. They can be conducted using packages such as `randomForest`.

Conclusions

Classification and regression trees are very popular in some disciplines – particularly those such as remote sensing that have access to enormous datasets. They are appealing because they are robust, can represent non-linear relationships, don't require that you preselect the variables to include in a model, and are easily interpretable.

However, regression trees can be problematic if they are over-fit to a dataset. Boosted forests and other extensions attempt to overcome some of the issues with (mostly univariate) regression trees, though require more computational power.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Breiman, L., J.H. Friedman, R.A. Olshen, and C.J. Stone. 1984. *Classification and regression trees*. Chapman & Hall, New York, NY.
- Buttrey, S.E., and L.R. Whitaker. 2015. treeClust: an R package for tree-based clustering dissimilarities. *Journal of Statistical Software* 7:227-236.
- Cutler, D.R., T.C. Edwards, Jr., K.H. Beard, A. Cutler, K.T. Hess, J. Gibson, and J.J. Lawler. 2007. Random forests for classification in ecology. *Ecology* 88:2783-2792.
- De'ath, G. 2002. Multivariate regression trees: a new technique for modeling species-environment relationships. *Ecology* 83:1105-1117.
- De'ath, G., and K.E. Fabricius. 2000. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology* 81:3178-3192.
- Elith, J., J.R. Leathwick, and T. Hastie. 2008. A working guide to boosted regression trees. *Journal of Animal Ecology* 77:802-813.
- Genuer, R., J-M. Poggi, and C. Tuleau-Malot. 2015. VSURF: an R package for variable selection using random forests. *Journal of Statistical Software* 7:19-33.
- Hothorn, T., K. Hornik, and A. Zeileis. 2006. Unbiased recursive partitioning: a conditional inference framework. *Journal of Computational and Graphical Statistics* 15:651-674.

Ouellette, M.-H., P. Legendre, and D. Borcard. 2012. Cascade multivariate regression tree: a novel approach for modelling nested explanatory sets. *Methods in Ecology and Evolution* 3:234-244.

Parisien, M.-A., and M.A. Moritz. 2009. Environmental controls on the distribution of wildfire at multiple spatial scales. *Ecological Monographs* 79:127-154.

Media Attributions

- trochosa_terricola_8313

33. Univariate Regression Trees

Learning Objectives

To understand how a univariate regression tree (URT) uses a set of explanatory variables to split a univariate response into groups.

To understand the importance of the complexity parameter (cp) table in evaluating a URT.

To build and interpret regression trees.

To explore how cross-validation allows assessment of how well a URT can predict the group identity of data that were not part of the tree's construction.

Readings (Recommended)

De'ath & Fabricius (2000)

Key Packages

```
require(tidyverse, mvpart, ggdendro)
```

Introduction

A univariate regression tree (URT) relates a single response variable to one or more explanatory variables. Many other techniques have this same overall goal; the difference for a URT is that:

- The dataset is divided via **binary splits** into mutually exclusive **nodes** (groups) such that the observations in each node are as similar as possible with respect to the response variable.
- Different explanatory variables can be used at each split.

The result is a tree structure in which each split divides the dataset into two groups.

Key Takeaways

A univariate regression tree (URT) evaluates multiple explanatory variables and identifies the value of one of those variables that separates sample units into two groups where the variance of a univariate response is minimized in those groups.

This process is then repeated **independently** for each group. The iterative or hierarchical nature of this process means that initial decisions can have cascading effects throughout the tree.

The URT Process

Groups are identified via binary splits in a tree structure: each split divides the dataset into two groups.

We begin by specifying the response variable and a number of potential explanatory variables. The response data are in a single node, called the **root node**. Therefore, this is a divisive approach – we seek to split the data into smaller groups.

A suite of potential explanatory variables are evaluated. These variables can be of any type – it is important that each explanatory variable is recognized as the correct class of object. We have seen numerous examples of categorical variables (e.g., male vs. female) and of continuous variables (e.g., elevation), but not many of ordinal variables (e.g., low / medium / high). For ordinal variables, the ordering of the levels is important and needs to be specified correctly so that the relative ordering of the levels is incorporated into the analysis. One way to do so is with the `ordered()` function. If an ordinal variable is not specified correctly, the tree could distinguish between potentially non-sensical combinations of levels (e.g., medium vs. low or high).

Each potential explanatory variable is compared to the response variable to identify:

- The value of the explanatory variable that would most strongly separate the response data into two nodes, and
- How well that explanatory variable would separate the data.

These decisions are made using the assumption that the goal is to identify nodes that are as homogeneous as possible with respect to the response variable. Homogeneity is assessed by calculating the **impurity** of the node: a completely homogeneous node contains zero impurity. Impurity can be calculated in different ways, depending on the characteristics of the response variable:

- Continuously-distributed: minimizing the within-node sums of squares (or, equivalently, maximizing the between-node sums of squares)
- Categorical: minimizing the misclassification rate

The explanatory variable that splits the root node into the most homogeneous pair of branches is selected. In other words, splitting on the basis of this variable at this value results in the lowest impurity. Each sample unit is assigned to one of the two nodes based on its value of the explanatory variable.

Now, the key step: each node is treated as an independent subset and the above procedure of

comparing explanatory variables and identifying the one that most strongly reduces the impurity of the resulting groups is repeated for each. **This process of treating nodes as independent subsets and analyzing them independently is why a URT can yield nonlinear solutions.**

All splits are optimal for the node and are evaluated without regard to other splits in the tree or the performance of the whole tree. However, the characteristics of the observations in a given node are conditional on earlier splits in the tree.

Splitting continues until one or more stopping criteria are satisfied. These stopping criteria include:

- A minimum number of observations that a node must contain to be split further. Nodes with fewer than this number of observations are called **leaves** or **terminal nodes**.
- A minimum number of observations that a terminal node must contain. For example, it generally would not be helpful to split a node such that a leaf contained a single observation.
- A minimum amount by which a split must increase the overall model fit in order to be performed. This amount is known as the **complexity parameter (cp)**.

Once the tree is grown, it is examined to determine whether it should be **pruned** back to a smaller and more desired size. The desired tree size can be established many ways, but often involves **cross-validation**. Cross-validation involves omitting one or more observations from the dataset, building a tree from the rest of the dataset, and then examining how well the tree predicts or classifies the omitted observation. The tree with the smallest predicted mean square error is the “best predictive tree” and should give the most accurate predictions.

The **fit** of a tree is assessed by calculating the proportion of the total sum of squares that it explains. Trees are summarized by their size (number of leaves) and overall fit (fraction of variance not explained by the tree). Fit can be defined by:

- **relative error** – total impurity of leaves divided by impurity of root node (i.e., undivided data), or the fraction of the total variance that is not explained by the tree. In other words, $1 - R^2$. Ranges from 0 to 1. Shown by open circles in Figure 3 (below, from De’ath 2002).
- **cross-validated relative error** – calculated based on the cross validations. Ranges from 0 (perfect predictor) to ~1 (poor predictor), and is always equal to or higher than that based on the full dataset. Shown by closed circles in Figure 3 below. The horizontal dashed line in the figures is 1 SE above the lowest cross-validated relative error.

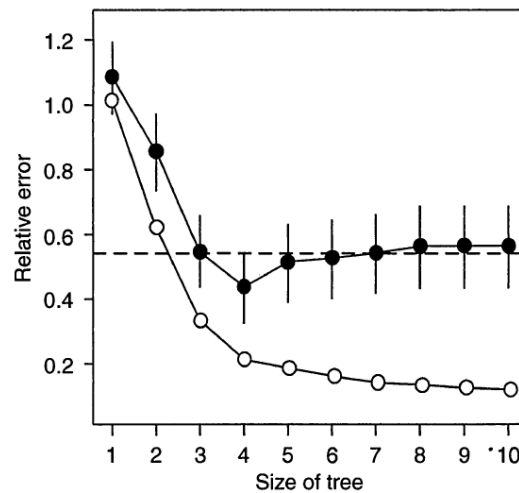


FIG. 3. Selection of the multivariate regression tree for the hunting spider data. The relative error (open circles) decreases with tree size, whereas the cross-validated relative error (filled circles) decreases to a minimum for a tree size of four, and then increases before flattening to a typical plateau. The vertical bars indicate one standard error for the cross-validated relative error, and the dashed line indicates one standard error above the minimum cross-validated relative error and suggests a tree size of four leaves.

For classification trees, the fit of the tree is assessed via misclassification rates. A **confusion matrix** is generated, where the rows are the observed classes and the columns are the predicted classes. Values along the diagonal are correctly classified while all others are misclassified.

A Worked URT Example

Let's illustrate the steps in a URT. We'll focus on one spider species, `troc.terr`, and one potential explanatory variable, `water`. See the introductory notes for instructions about loading these data. We'll do some illustrative computations, not all possible comparisons.

Decisions about how to split the data will be made on the basis of the variance within the resulting groups. We can calculate the total variance using Huygens' theorem (recall this from the notes about PERMANOVA?): the sum of squared differences between points and their centroid is equal to the sum of the squared interpoint distances divided by the number of points.

Since I'm going to calculate variances for different splits of the data below, I'll create a function to do so.

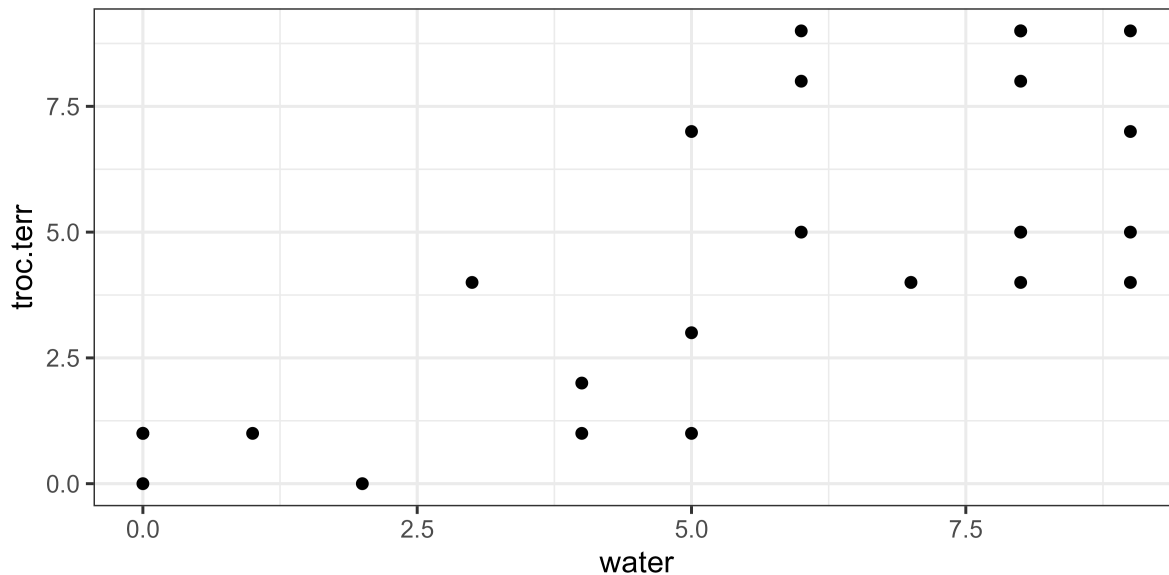
```
calculate.variance <- function(object, value) {
  object %>%
  mutate(breakpoint = if_else(water < value, "low", "hi")) %>%
  group_by(breakpoint) %>%
  summarize(N = length(troc.terr),
    variance = sum(dist(troc.terr)^2) / N,
    mean = mean(troc.terr))
}
```

Note that this is hard-coded with `water` as the explanatory variable and `troc.terr` as the response

variable. We specify the name of the object containing these variables and the value of water at which we want to split the data. This function is available in the GitHub repository.

Root Node

Here's a simple scatterplot of the data.



Summarizing the data in the root node:

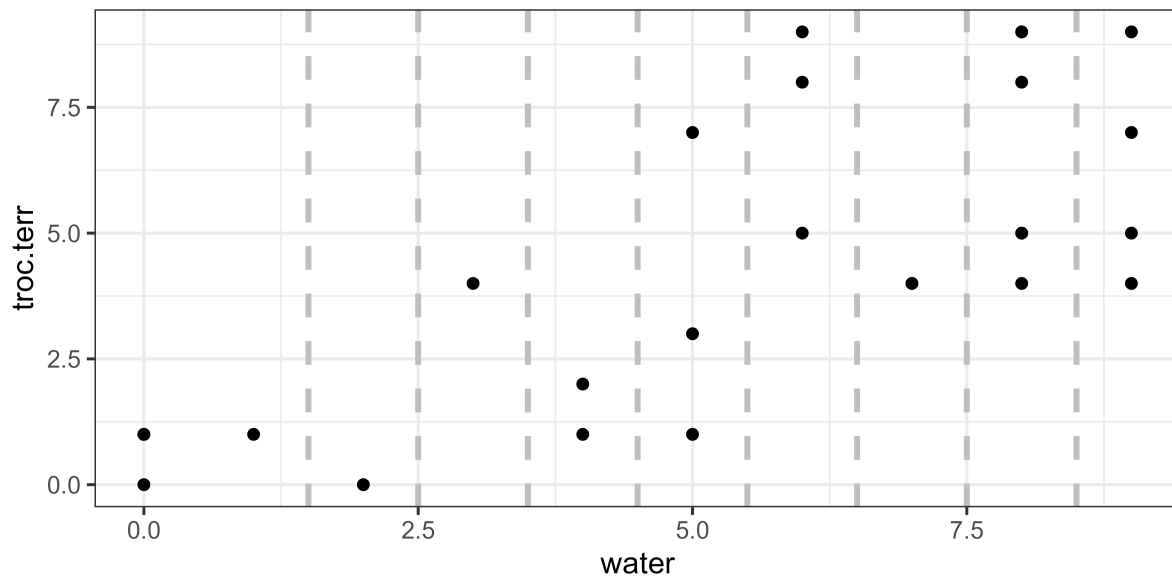
```
calculate.variance(object = spider, value = NA)
```

```
# A tibble: 1 × 4  
  breakpoint    N variance  mean  
  <chr>      <int>    <dbl> <dbl>  
1 NA         28      263   4.5
```

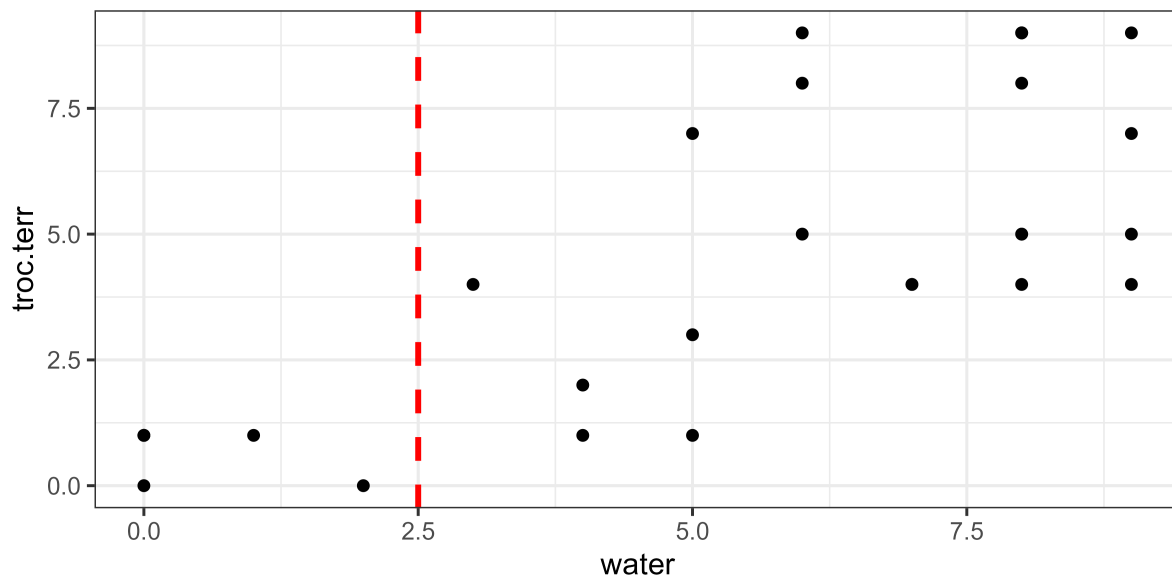
The total variance within the dataset is 263, and the average spider abundance is 4.5. Note that this calculation doesn't depend on the explanatory variable at all.

Identifying First Binary Split

Now, let's divide the data into two groups on the basis of the explanatory variable. There are many ways we could do so, but any number between two unique values of water will give the same answer – for example, imagine drawing a vertical line below at 2.1 or 2.5 or 2.9. This means we can simply focus on the midpoints between the unique values, shown by the dashed vertical lines here:



Each of these lines would be evaluated separately. For example, the line at water = 2.5 distinguishes sample units with water below this value from sample units above this value:



How would this break point affect the variation within the associated groups? We can use Huygens' theorem again, applying it separately to each group:

```
calculate.variance(object = spider, value = 2.5)
```

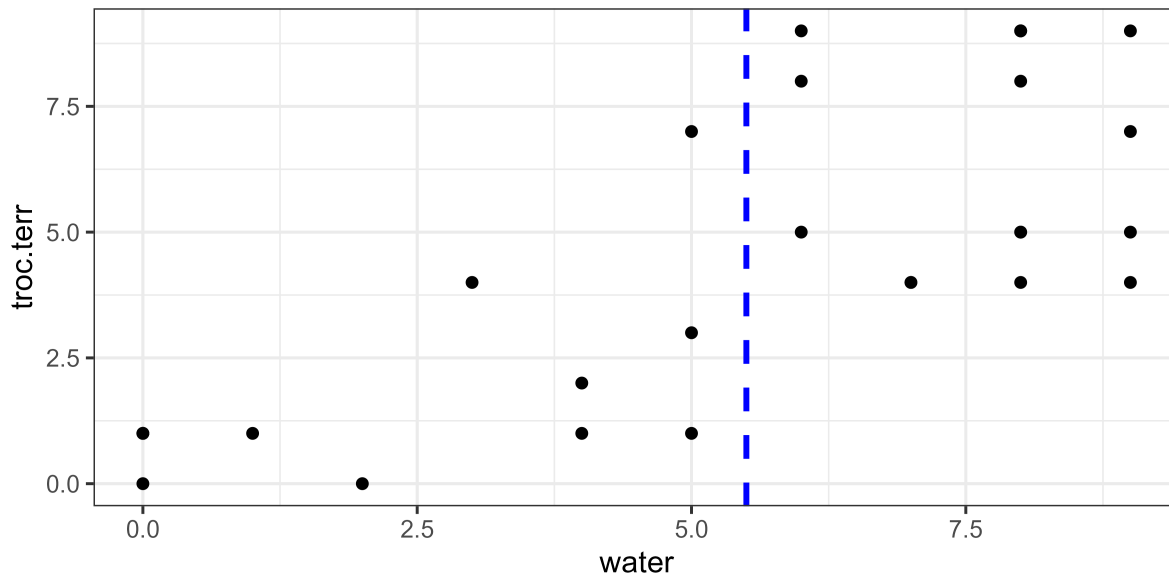
```
# A tibble: 2 × 4
```

	breakpoint	N	variance	mean
	<chr>	<int>	<dbl>	<dbl>
1	hi	22	149.	5.55
2	low	6	1.33	0.667

The total variance within these groups is $149 + 1.33 = 150.33$. The difference between this and the total variance in the dataset (263; see root node above) is the variance between the two groups. There are 22 plots in the 'hi' group and they have a mean spider abundance of 5.55. There are 6 plots in the 'low' group and they have a mean abundance of 0.67.

Let's consider a few other break points.

Water = 5.5:

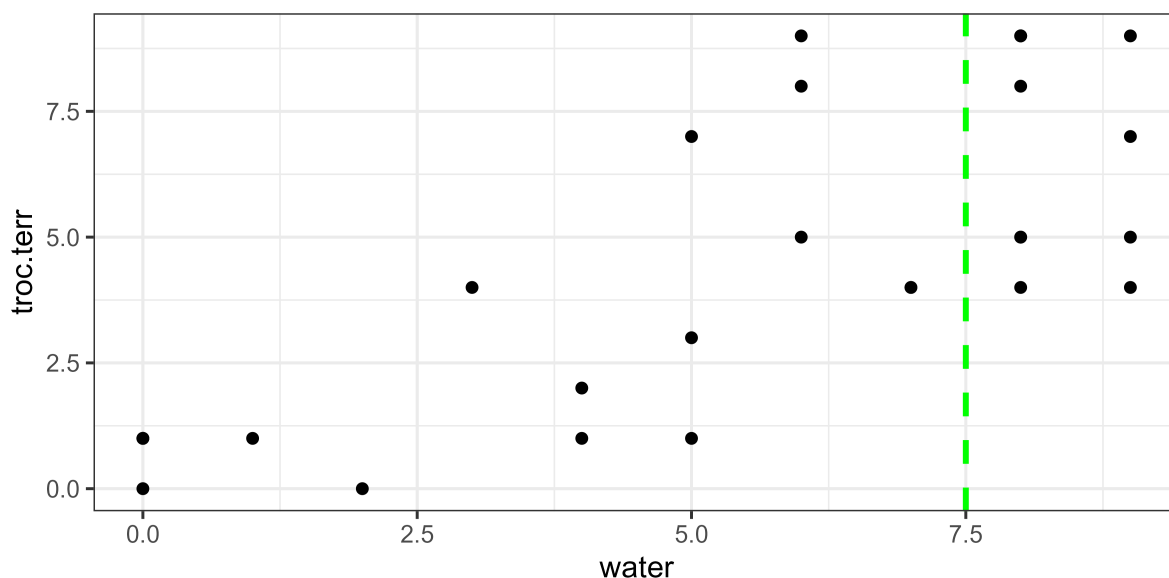


```
calculate.variance(object = spider, value = 5.5)
```

```
# A tibble: 2 × 4
  breakpoint     N variance  mean
  <chr>      <int>   <dbl> <dbl>
1 hi         16     70    6.5
2 low        12    43.7   1.83
```

Total variance within groups = $70 + 43.7 = 113.7$.

Water = 7.5:



```
calculate.variance(object = spider, value = 7.5)
```

```
# A tibble: 2 × 4
  breakpoint      N variance  mean
  <chr>      <int>    <dbl> <dbl>
1 hi         11     46.2  6.73
2 low        17    127.  3.06
```

Total variance within groups = $46.2 + 127 = 173.2$.

Summarizing this information:

Break point	Variance Within Groups
None (i.e., Total)	263
water = 2.5	150.33
water = 5.5	113.7
water = 7.5	173.2

Of the three potential break points that we calculated, the lowest within-group variance is produced when water = 5.5. Therefore, this is the one that we would use to form the first binary split.

Using this break point, the unexplained variance is 113.7. We can also express this as a proportion of the total variance calculated above: $113.7 / 263 = 0.432$. This is the **relative error** associated with this split.

Note 1: I illustrated this approach by testing three potential splits of water, but operationally a URT will evaluate all possible splits. You can use the R functions below to verify that if water is the only potential explanatory variable specified, the optimal break point is water = 5.5.

Note 2: I illustrated this approach by testing one potential explanatory variable, but operationally a URT will evaluate all specified potential explanatory variables. Whether the split occurs on the basis of water will depend on which other potential explanatory variables are in the model, and how well the optimal splits for those variables minimize impurity within the groups.

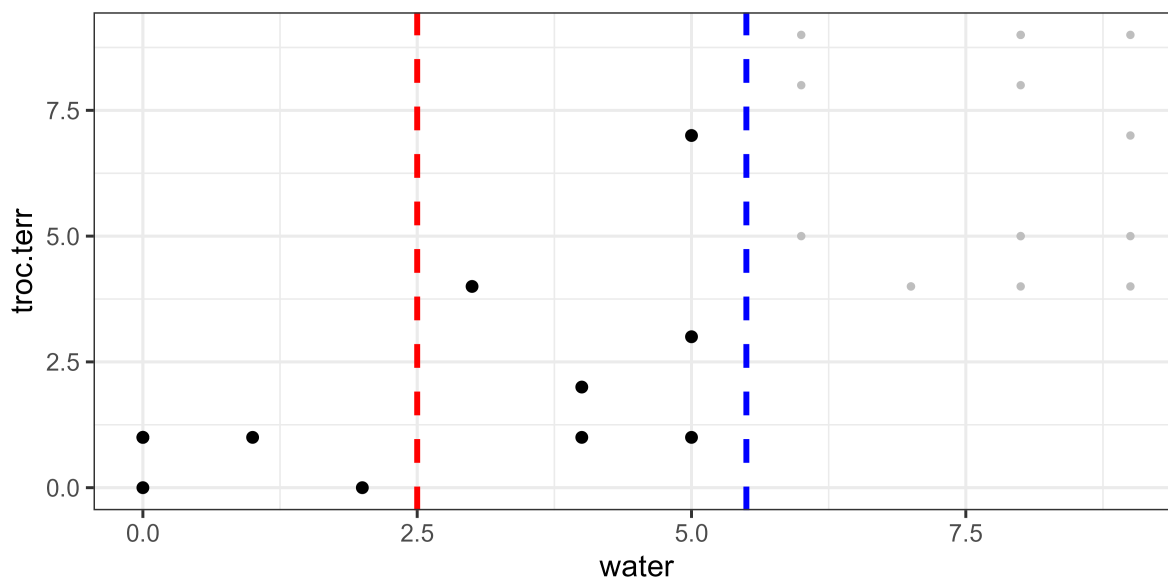
Code to create above figures of various break points (comment out lines as desired):

```
ggplot(data = spider, aes(x = water, y = troc.terr)) +  
  geom_point() +  
  geom_vline(xintercept = c(1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5),  
    col = "grey", linetype = "dashed", linewidth = 1) +  
  geom_vline(xintercept = 2.5, col = "red",  
    linetype = "dashed", linewidth = 1) +  
  geom_vline(xintercept = 5.5, col = "blue",  
    linetype = "dashed", linewidth = 1) +  
  geom_vline(xintercept = 7.5, col = "green",  
    linetype = "dashed", linewidth = 1)+  
  theme_bw()
```

Identifying Second Binary Split

Now, the process is repeated with one of the groups created from the above binary split. Let's start with the group defined by $\text{water} < 5.5$, ignoring observations with $\text{water} \geq 5.5$. The summary of this split above indicates that there are 12 observations with $\text{water} < 5.5$. How can we most strongly separate these 12 observations into two groups?

Again, all possible splits are evaluated. For illustrative purposes, we'll just consider one possible split of these data – $\text{water} = 2.5$:



In this graphic, the group of points are to the left of the dashed blue line; the values above this line are shown as small grey points because they are excluded from all calculations with respect to this

binary split. The dashed red line is the possible break point at 2.5. Note that this is the same break point that we saw above when considering the first split binary split; the group to the left of this is unchanged but the group to its right now includes fewer values.

```
spider %>%
  filter(water < 5.5) %>%
  calculate.variance(value = 2.5)
```

```
# A tibble: 2 × 4
  breakpoint      N variance  mean
  <chr>      <int>    <dbl> <dbl>
1 hi         6      26      3
2 low        6      1.33  0.667
```

The total variance within these groups = $26 + 1.33 = 27.33$.

Again, all possible values of the binary split of this group are assessed and the split that produces the smallest within-group variation is identified. If there were multiple potential explanatory variables, they would each be evaluated using this subset of the full dataset.

If we selected this break point to split the data, then the remaining variance within groups would be calculated as the variance within these groups plus the variance within the other group from the first split ($\text{water} \geq 5.5$; variance = 70; see above). Therefore, the total variance within groups is $26 + 1.33 + 70 = 97.33$, and the relative error has decreased to $97.33 / 263 = 0.370$.

Code to create above figure:

```
ggplot(data = spider, aes(x = water, y = troc.terr)) +
  geom_point(size = 0.75, colour = "grey") +
  geom_point(data = spider[spider$water < 5.5, ]) +
  geom_vline(xintercept = 5.5, col = "blue",
    linetype = "dashed", size = 1) +
  geom_vline(xintercept = 2.5, col = "red",
    linetype = "dashed", size = 1) +
  theme_bw()
```

Identifying Third Binary Split, and Beyond

For the third split, we focus on the data from the group with $\text{water} \geq 5.5$ and determine which of the possible splits of these data will most strongly separate them into two groups. I haven't shown this process here; it's the same as we saw for the second split. Again, the data from the other group ($\text{water} < 5.5$) are not part of this calculation.

This process repeats for all subsequent groups. The process stops when one or more of the stopping criteria are satisfied.

Regression Trees in R

There are several packages that will conduct regression trees in R, including `rpart` (part of the base installation), `mvpart`, `tree`, and `party` (Hothorn et al. 2006).

We'll focus on `mvpart`. See previous chapter for details about how to install `mvpart` from a github archive.

In addition to the functions used below, `mvpart` contains several other useful functions, including:

- `scaler()` – various standardization methods.
- `gdist()` – calculates dissimilarities. Bray-Curtis distances are the default.
- `xdiss()` – calculates 'extended dissimilarities' (De'ath 1999). Meant for datasets that span long gradients and therefore contain sample units that do not have species in common. This is comparable to the `stepacross()` function from `vegan`.

`mvpart::mvpart()`

The function we will use is called `mvpart()`. This function contains lots of arguments because it is a 'wrapper' function: it automatically calls several other functions during its execution. In particular, it calls `rpart()`, the work-horse function that actually does the regression tree, as well as functions to conduct cross-validations, plot the data, etc. Its usage is:

```
mvpart(form,
  data,
  minauto = TRUE,
  size,
  xv = c("lse", "min", "pick", "none"),
  xval = 10,
  xvmult = 0,
  xvse = 1,
  snip = FALSE,
  plot.add = TRUE,
  text.add = TRUE,
  digits = 3,
  margin = 0,
  uniform = FALSE,
  which = 4,
  pretty = TRUE,
  use.n = TRUE,
  all.leaves = FALSE,
  bars = TRUE,
  legend,
  bord = FALSE,
  xadj = 1,
  yadj = 1,
  prn = FALSE,
  branch = 1,
  rsq = FALSE,
  big.pts = FALSE,
  pca = FALSE,
  interact.pca = FALSE,
```

```
wgt.ave.pca = FALSE,
keep.y = TRUE,
...
)
```

Note that most arguments have default values. The key arguments include:

- `form` – the formula to be tested. A URT is performed if the left-hand side of the formula is a single variable, otherwise a MRT is performed. See notes about the `method` argument below.
- `data` – the data frame containing the variables named in the right-hand side of the formula.
- `size` – the size of the tree to be generated
- `xv` – selection of tree by cross-validation. Options:
 - `1se` – best tree within one SE of overall best. A commonly used option. Note that this is ‘one SE’, not ‘el SE’.
 - `min` – best tree
 - `pick` – pick tree size interactively. Returns a graph of relative error and cross-validated relative error as a function of tree size, and allows you to choose the desired tree size from this graph. See ‘Cross-Validation’ below.
 - `none` – no cross-validation
- `xval` – number of cross-validations to perform. Default is 10. Borcard et al. (2018) describe it as indicating how many groups to divide the data into. There appears to be some uncertainty about how this argument works; see ‘Cross-Validations’ below.
- `xvmult` – number of ‘multiple cross-validations’ to perform. Default is 0.
- `plot.add` – plot the tree? Default is `TRUE` (yes).
- `snip` – interactively prune the tree? Default is `FALSE` (no).
- `all.leaves` – annotate all nodes? Default is `FALSE` (no), which just annotates terminal nodes.

The usage of `mvpart()` ends with ‘...’. This means that you can also include arguments from other functions (those that `mvpart()` calls internally) without calling those functions directly. For example, the `rpart()` function contains some additional important arguments:

- `method` – type of regression tree to be conducted. If this argument is not specified, the function guesses at the appropriate option based on the class of the response variable or matrix. The options are:
 - `anova` – for a ‘classical’ URT. Assumed default for all data that do not meet specifications for other methods.
 - `poisson` – for a URT with a response variable that contains count data.
 - `class` – for a URT classification tree. Assumed default if data are of class factor or class character.
 - `exp` – for a URT with a response variable that contains survival data.
 - `mrt` – for a MRT. Assumed default if data are of class matrix.
 - `dist` – for a MRT. Assumed default if response is a distance matrix.
- `dissim` – how to calculate sum of squares. Only used for `method = "anova"` or `method = "mrt"`. Options are
 - `euclidean` – sum of squares about the mean
 - `manhattan` – aka city block; sum of absolute deviations about the mean

Additional controls are available through the `rpart.control()` function, including:

- `minsplit` – the minimum number of observations that must exist in a node for it to be split further. Default is 5.
- `minbucket` – minimum number of observations in any terminal node. Default is one-third of `minsplit` (i.e., `round(minsplit/3)`).

- `cp` – complexity parameter; the amount a split must increase the fit to be undertaken. Default is 0.01.
- `xval` – number of cross-validations. Default is to split the data into 10 groups.
- `maxcompete` – number of competing variables to consider at each node. A competing variable is one that would form the basis for the split if the primary variable was not present in the model. Default is 4.
- `maxsurrogate` – number of surrogate variables to consider at each node. A surrogate is a variable that would be used if the primary variable was missing data for a given sample unit. Default is 0.

These can generally be called directly within `mvpart()`, but for more explicit control they can be called by including the `control` argument in `mvpart()`. Examples of this are provided below.

The function returns an object of class ‘`rpart`’ with numerous components. Key components are:

- `frame` – a dataframe containing the summary information about the tree. There is one row per node, and the columns contain the variable used in the split, the size of the node, etc. The terminal leaves are indicated by `<leaf>` in this column.
- `where` – the leaf node that each observation falls into.
- `cptable` – table of optimal prunings based on a complexity parameter.

See `?rpart.object` for more information.

Sample URTs in R

T. terricola Abundance and Water

Let’s begin with a single explanatory variable as in our worked example above. We will relate the abundance of *T. terricola* (`troc.terr`) to the abundance of water (`water`). To begin, we’ll turn off cross-validation and the automatic plotting of the results, and focus on the numerical summaries:

```
Troc.terr.water <- mvpart(spider[, "troc.terr"] ~ water,
  data = env,
  xv = "none",
  plot.add = FALSE,
  control = rpart.control(xval = 0))
```

A numerical summary of the tree can be viewed by calling the object directly:

```
Troc.terr.water
```

```
n= 28

node), split, n, deviance, yval
  * denotes terminal node

1) root 28 263.000000 4.5000000
```



```

2) water< 5.5 12 43.666670 1.8333330
4) water< 2.5 6 1.333333 0.6666667 *
5) water>=2.5 6 26.000000 3.0000000
10) water< 4.5 3 4.666667 2.3333330 *
11) water>=4.5 3 18.666670 3.6666670 *
3) water>=5.5 16 70.000000 6.5000000
6) water>=6.5 13 58.769230 6.3076920
12) water< 7.5 2 0.000000 4.0000000 *
13) water>=7.5 11 46.181820 6.7272730 *
7) water< 6.5 3 8.666667 7.3333330 *

```

Each line in this summary shows:

- Node (numbered in order created). Each split automatically creates two additional nodes. For example, the first split creates nodes 2 and 3, the second split creates nodes 4 and 5, etc.
- Explanatory variable used to create split, and value at which split occurs
- Number of observations in node
- Deviance (variation within node)
- Mean value of response variable in node

For example, node 1 is the root node and contains all 28 observations. The total variance is 263, and the mean abundance of the spider is 4.5. These values are as reported in our worked example.

The first split was made on the basis of water < 5.5 (node 2) or >= 5.5 (node 3). The associated values reported for each of these nodes are as reported in our worked example.

The second split is made on the basis of water < 2.5 (node 4) or >= 2.5 (node 5). These values are also as reported in our worked example.

The results of subsequent splits are also reported. Again, each is created by focusing on a subset of the data and then evaluating all possible splits of that subset.

***T. terricola* Abundance and Multiple Potential Explanatory Variables**

What if we include more than one potential explanatory variable? Once again, we'll focus on the numerical summaries and therefore turn off cross-validation and the automatic plotting of the results.

```

Troc.terr <- mvpart(spider[, "troc.terr"] ~ .,
  data = env,
  xv = "none",
  plot.add = FALSE,
  control = rpart.control(xval = 0))

```

Note that the formula includes a period ('.') in the right-hand side of the equation – this indicates that all columns within the object containing the explanatory variables (`env`) should be used. If desired, we could have typed out all of the variable names, including them as main effects without interactions.

A numerical summary of the tree can be viewed by calling the object directly:

```
Troc.terr
```

```

n= 28

node), split, n, deviance, yval
  * denotes terminal node

1) root 28 263.0000000 4.500000
 2) herbs< 8.5 20 79.8000000 2.900000
   4) reft>=6 10 6.9000000 1.100000
     8) herbs< 6.5 8 1.5000000 0.750000 *
     9) herbs>=6.5 2 0.5000000 2.500000 *
   5) reft< 6 10 8.1000000 4.700000 *
 3) herbs>=8.5 8 4.0000000 8.500000
   6) reft>=7.5 2 0.5000000 7.500000 *
   7) reft< 7.5 6 0.8333333 8.833333 *
```

This summary is in the same format as above. After evaluating all of the potential explanatory variables, it was determined that herbs most strongly split the data into groups – this means that it did better than water which we had considered alone earlier. Furthermore, different variables have been selected in different splits: the first split was made on the basis of herbs, the second split on the basis of reft (light), etc.

More detailed information is available using `summary()`:

```
summary(Troc.terr)
```

```

Call:
mvpert(form = spider[, "troc.terr"] ~ ., data = env, xv = "none",
  plot.add = FALSE, control = rpart.control(xval = 0))
n= 28

      CP nsplit  rel error
1 0.68136882    0 1.00000000
2 0.24638783    1 0.31863118
3 0.01863118    2 0.07224335
4 0.01013942    3 0.05361217
5 0.01000000    4 0.04347275

Node number 1: 28 observations,    complexity param=0.6813688
mean=4.5, MSE=9.392857
left son=2 (20 obs) right son=3 (8 obs)
Primary splits:
  herbs < 8.5 to the left, improve=0.6813688, (0 missing)
  water < 5.5 to the left, improve=0.5678074, (0 missing)
  moss < 7.5 to the right, improve=0.4850760, (0 missing)
  sand < 4 to the right, improve=0.3737131, (0 missing)
  reft < 7.5 to the right, improve=0.3417377, (0 missing)

Node number 2: 20 observations,    complexity param=0.2463878
mean=2.9, MSE=3.99
left son=4 (10 obs) right son=5 (10 obs)
Primary splits:
  reft < 6 to the right, improve=0.8120301, (0 missing)
```

```

    water < 5.5 to the left, improve=0.7230450, (0 missing)
    twigs < 3.5 to the left, improve=0.7230450, (0 missing)
    moss < 6 to the right, improve=0.6562112, (0 missing)
    sand < 5.5 to the right, improve=0.3516759, (0 missing)

Node number 3: 8 observations, complexity param=0.01013942
mean=8.5, MSE=0.5
left son=6 (2 obs) right son=7 (6 obs)
Primary splits:
    reft < 7.5 to the right, improve=0.6666667, (0 missing)
    water < 7 to the left, improve=0.3000000, (0 missing)
    moss < 1.5 to the right, improve=0.3000000, (0 missing)

Node number 4: 10 observations, complexity param=0.01863118
mean=1.1, MSE=0.69
left son=8 (8 obs) right son=9 (2 obs)
Primary splits:
    herbs < 6.5 to the left, improve=0.71014490, (0 missing)
    sand < 2.5 to the right, improve=0.50310560, (0 missing)
    water < 3 to the left, improve=0.40821260, (0 missing)
    reft < 8.5 to the right, improve=0.11663220, (0 missing)
    moss < 8.5 to the right, improve=0.01449275, (0 missing)

Node number 5: 10 observations
mean=4.7, MSE=0.81

Node number 6: 2 observations
mean=7.5, MSE=0.25

Node number 7: 6 observations
mean=8.833333, MSE=0.1388889

Node number 8: 8 observations
mean=0.75, MSE=0.1875

Node number 9: 2 observations
mean=2.5, MSE=0.25

```

The `summary()` function reports more information about each split, including the value at which each potential explanatory variable would optimally split the data and how much of an improvement doing so would make to the fit. For example, look at the information for the first split, repeated here:

```

Node number 1: 28 observations, complexity param=0.6813688
mean=4.5, MSE=9.392857
left son=2 (20 obs) right son=3 (8 obs)
Primary splits:
    herbs < 8.5 to the left, improve=0.6813688, (0 missing)
    water < 5.5 to the left, improve=0.5678074, (0 missing)
    moss < 7.5 to the right, improve=0.4850760, (0 missing)
    sand < 4 to the right, improve=0.3737131, (0 missing)
    reft < 7.5 to the right, improve=0.3417377, (0 missing)

```

Making this split on the basis of herbs improved the fit by 0.681. If this first split had been made on the basis of water, the split would have separated observations with water < 5.5 from water ≥ 5.5 – just like we saw above – and would have improved the fit by 0.568 (i.e., 1 – relative error of 0.432 from above). The latter split would have occurred, for example, if herbs was not included in this list of explanatory variables. Note, however, that making the first split on the basis of any other variable would have changed the rest of the tree as the optimal separation differs among variables. Thus, **splits are always conditional on all preceding branches in the tree.**

We will use this tree to illustrate additional aspects of URTs.

Complexity Parameter (cp) Table

Included near the top of the summary is the complexity parameter (cp) table. This table is available as an element within the object (`Troc.terr$cp`) and can also be printed or graphed directly.

```
printcp(Troc.terr)
```

```
Regression tree:
mvpert(form = spider[, "troc.terr"] ~ ., data = env, xv = "none",
  plot.add = FALSE, control = rpart.control(xval = 0))

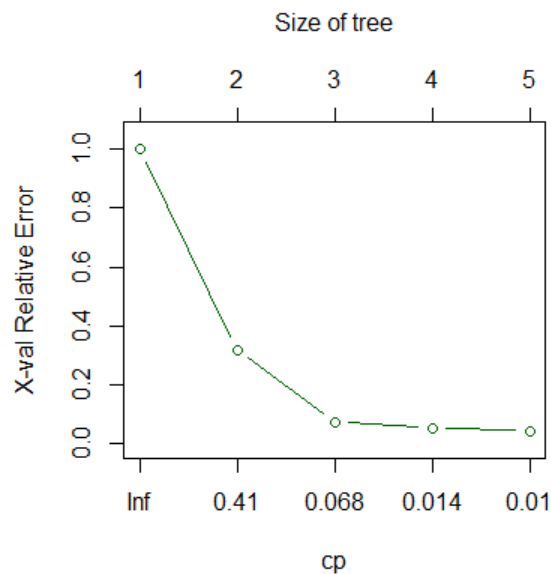
Variables actually used in tree construction:
[1] herbs reft

Root node error: 263/28 = 9.3929

n= 28
```

	CP	nsplit	rel error
1	0.681369	0	1.000000
2	0.246388	1	0.318631
3	0.018631	2	0.072243
4	0.010139	3	0.053612
5	0.010000	4	0.043473

```
plotcp(Troc.terr)
```



*Graphical summary of the complexity parameter table for the *Trochosa terricola* univariate regression tree.*

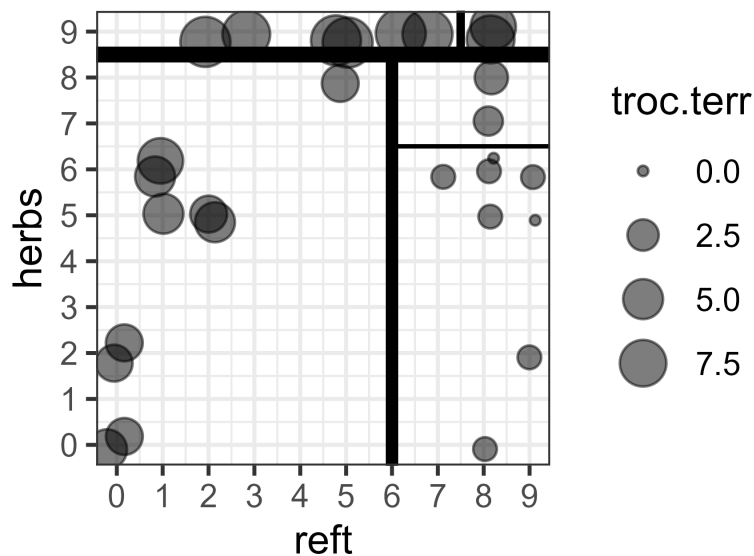
The graph of the cp table shows error as a function of cp (bottom axis) and, equivalently, tree size (top axis). Note that the horizontal axis is scaled to reflect consistent changes in tree size but that cp changes by very different amounts as tree size increases. Although the y-axis is labeled for cross-validated relative error, in this case it is only showing the relative error (indicated by the green color; see 'Cross-Validation' below for the alternative). This shows the entire tree, without any pruning.

Groups From a URT

Working Through the Tree

To determine which group a given sample is classified into, you could work your way through the tree, answering the question posed at each node.

Another way to explore the data is to overlay the splits onto a scatterplot. Although we allowed the tree to consider six explanatory variables, only two were selected in the above tree. Therefore, we can create a simple scatterplot of these two variables. We will show the abundance of *troc.terr* by adjusting the size of the symbols.



Visual representation of the univariate regression tree for *Trochosa terricola*. Symbol size is proportional to abundance of this spider. Solid black lines indicate the binary splits selected in this tree, with the thickest line representing the first split and the narrowest line the last split.

Notes about this graphic:

- The abundance data are jittered slightly to show duplicates.
- Each split is represented by a break within this space. The first split is the thickest line and the only line to span one of the axes. Each subsequent split is shown by a thinner line, and occurs within a previous split.
- Each of the 'boxes' defined by these splits represents a group of sample units with similar environmental conditions and abundances of this hunting spider.

(here is the code to create this figure)

```
ggplot(data = spider, aes(x = reft, y = herbs)) +
  scale_y_continuous(breaks = 0:9) +
  scale_x_continuous(breaks = 0:9) +
  coord_cartesian(xlim = c(0, 9), ylim = c(0, 9)) +
  geom_point(aes(size = troc.terr), alpha = 0.5,
    position = position_jitter(width = 0.25, height = 0.25)) +
  geom_hline(aes(yintercept = 8.5),
    linewidth = 2) + # split 1
  geom_segment(aes(x = 6, y = -1, xend = 6, yend = 8.5),
    linewidth = 1.5) + # split 2
  geom_segment(aes(x = 7.5, y = 8.5, xend = 7.5, yend = 10),
    linewidth = 1) + # split 3
```

```
geom_segment(aes(x = 6, y = 6.5, xend = 10, yend = 6.5),
  linewidth = 0.5) + # split 4
theme_bw() + theme(aspect.ratio = 1)

ggsave("graphics/URT.png", width = 3, height = 3, units = "in", dpi = 600)
```

Key Takeaways

A URT can be read as a series of questions that, when answered, predict the group identity of a new observation. The questions relate to the selected explanatory variables, and the prediction is the mean value of the response variable.

Indexing Leaves

An integer coding for the terminal node to which each sample unit was assigned is stored in the resulting object:

```
Troc.terr$where
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
6  6  6  6  6  6  9  6  6  9  6  9  9  9  9  8  8  4  5  6  5
22 23 24 25 26 27 28
4  4  4  4  4  4  4
```

Note that these groups are terminal node numbers; they do not include nodes that were subsequently divided in the tree (e.g., 1, 2, 3).

We can create an index from this information and then use that index to manipulate the data. For example, let's view the environmental variables for the group identified by terminal node 6:

```
groups.urt <- data.frame(group = Troc.terr$where)
```

```
env[which(groups.urt == 6) , ]
```

```
   water sand moss reft twigs herbs
1      9    0    1    1     9     5
2      7    0    3    0     9     2
3      8    0    1    0     9     0
4      8    0    1    0     9     0
5      9    0    1    2     9     5
6      8    0    0    2     9     5
8      6    0    2    1     9     6
```

9	7	0	1	0	9	2
11	9	5	5	1	7	6
20	3	7	2	5	0	8

These are the values of the environmental variables for the observations assigned to node 6.

Plotting a Regression Tree

To plot a regression tree, we can include the default argument of `'plot.add = TRUE'` in our call of `mvpart()` and/or we can plot it separately.

Plotting During Analysis

Plotting the tree directly while conducting the analysis provides access to some other optional arguments and automatically includes some other information, such as the relative error reported below the tree.

```
Troc.terr <- mvpart(spider[, "troc.terr"] ~ .,
  data = env,
  xv = "none",
  plot.add = TRUE,
  all.leaves = TRUE,
  control = rpart.control(xval = 0))
```




- The lengths of the vertical branches of the tree are proportional to the amount of variation explained by that split (analogous to how the vertical axis in a dendrogram is proportional to the amount of variation explained by a cluster).
- At each intermediate node, the variable selected to split the dataset is shown, along with the value at which it splits it.
- At leaves (terminal nodes), the numbers are the mean value of the response (abundance of *T. terricola* in this case) and the number of sample units in that group.

Here is an example of how to draw a regression tree using `ggplot2` and `ggdendro`. We already loaded `ggplot2` as part of `tidyverse`; we now need to (possibly install and then) load `ggdendro`:

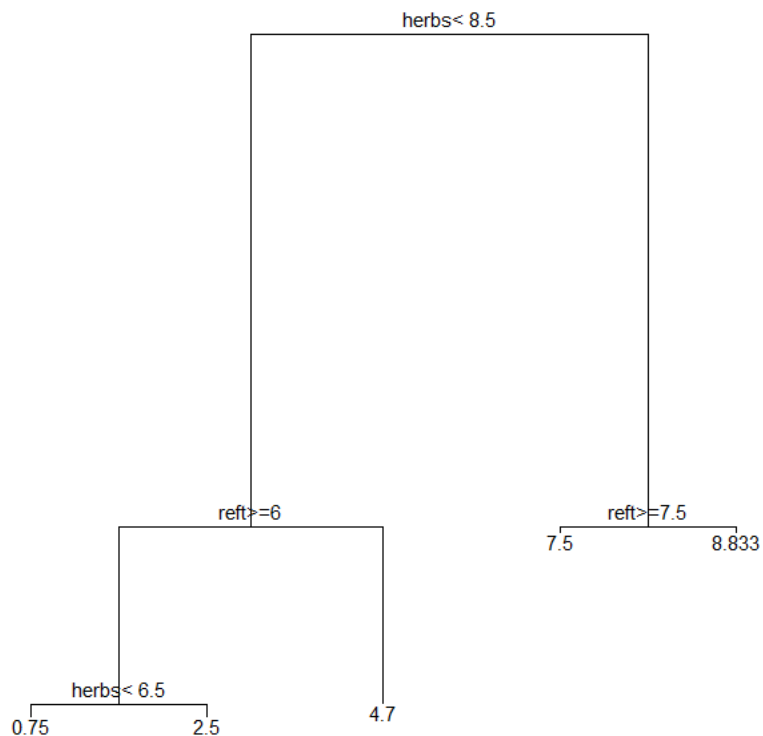
The `ggdendro` package includes a function, `dendro_data()`, that organizes the components of a dendrogram in an object of class 'dendro':

By inspecting this object, you can see that it contains three key sub-objects:

- `segments` – the coordinates for drawing the tree itself.
- `labels` – the data to report at each non-leaf node, and the coordinates of each such node.
- `leaf_labels` – the data to report at each leaf (terminal node), and the coordinates of each leaf.

To draw the tree, we will call each sub-object within a separately line of code:

```
ggplot() +
  geom_segment(data = ddata$segments,
    aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_text(data = ddata$labels,
    aes(x = x, y = y, label = label), size = 4, vjust = -0.5) +
  geom_text(data = ddata$leaf_labels,
    aes(x = x, y = y, label = label), size = 4, vjust = 1) +
  theme_dendro()
```



*Simple version of univariate regression tree of *Trochosa terricola*, as drawn in `ggdendro`.*

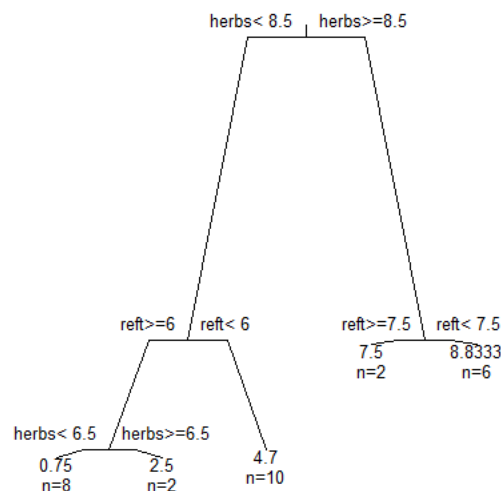
Once you have a base graph like this, you can use the other capabilities of `ggplot2` to customize it as desired.

This code can also be easily modified for application to dendrograms from hierarchical cluster analyses.

Plotting Using Base R Capabilities

Using the base R plotting functionality to plot the tree, we plot the skeleton of the tree first and then add the textual labels to it. We can also customize many elements of the plot, such as the angle of the branches:

```
plot(Troc.terr, branch = 0.5, margin = 0.1)
text(Troc.terr, use.n = TRUE, cex = 0.8)
```



*Simple version of univariate regression tree of *Trochosa terricola*, as drawn with base R plotting capabilities and altered branch angles.*

Arguments to customize the skeleton of the tree (i.e., `plot()`) include:

- `branch` – shape of branches; ranges from 1 (square-shouldered branches) to 0 (V-shaped branches).
- `margin` – amount of white space to leave around borders of tree. Helpful if you have long labels.
- `uniform` – we didn't use this here, but setting it `TRUE` results in a tree in which the vertical branches are not proportional to the amount of variation explained. Also applies to the `text()` function.

Arguments to customize the textual labels (i.e., `text()`) include:

- `use.n` – whether to include the sample size in each leaf.
- `cex` – amount to expand characters. Values `< 1` make the text smaller.
- `all.leaves` – whether to report the mean value and sample size in every node (`TRUE`) or just in

the leaves (`FALSE`).

Other options also exist; see `?plot.rpart` and `?text.rpart` for details.

Cross-Validation

Theory and Considerations

The above tree provided an introduction to the overall structure of these objects. However, it may not be optimally sized. Classification and regression trees are overly 'optimistic' in their ability to classify objects, and thus it is preferable to test trees using data that were not part of their construction. This process is called **cross-validation**. Specifically, cross-validation involves:

- Omitting a subset of the data
- Building a tree from the rest of the dataset
- Examining how well the tree predicts or classifies the omitted subset

Model fit is expressed as relative error, as above. The tree with the smallest cross-validated error should give the most accurate predictions. However:

- Cross-validated relative error cannot be smaller than the relative error for a tree of a given size
- While relative error decreases as the tree gets more complex, this is not necessarily the case for cross-validated relative error.

There are three aspects of cross-validation to consider.

1. **How many groups to split the data into?** The default is `xval = 10`, but if the the dataset is small, Borcard et al. (2018) suggest using as many groups as there are samples in the dataset. The help file indicates that `xval` can be specified as an argument directly within `mvpart()` or within `rpart.control()`. However, my testing indicates that calling it directly in `mvpart()` does not alter the results (compare `x$control$xval` of model `x` with and without this argument). I therefore recommend calling it in `rpart.control()` as shown below.
2. **How to assess the consistency of the cross-validation process itself?** This can be done by repeating the cross-validation procedure a specified number of times (`xvmult`). The default is `xvmult = 0`. Note that it appears to me that this is only helpful if you have used fewer cross-validations than are possible (i.e., than there are samples in the dataset). You can explore this by altering the settings below.
3. **How to use the results of the cross-validation.** We can either allow the function to automatically choose the solution (`xv = "min"` or `xv = "1se"`; see definitions above), or interactively choose it ourselves (`xv = "pick"`). When doing this interactively, an overly large tree is grown – i.e., the tree is overfit – and the complexity parameter (`cp`) is calculated for each tree size. See below for an example and explanation of the resulting figure.

If you use fewer cross-validations than are possible from your dataset, the optimal size of tree can vary from run to run. One way to make your results reproducible – generating the same solution each time – is to use `set.seed()` to set the random number generator immediately before running `mvpart()`.

Key Takeaways

Cross-validation is critical for assessing how well a URT can classify (predict the group identity) of new observations.

Illustration

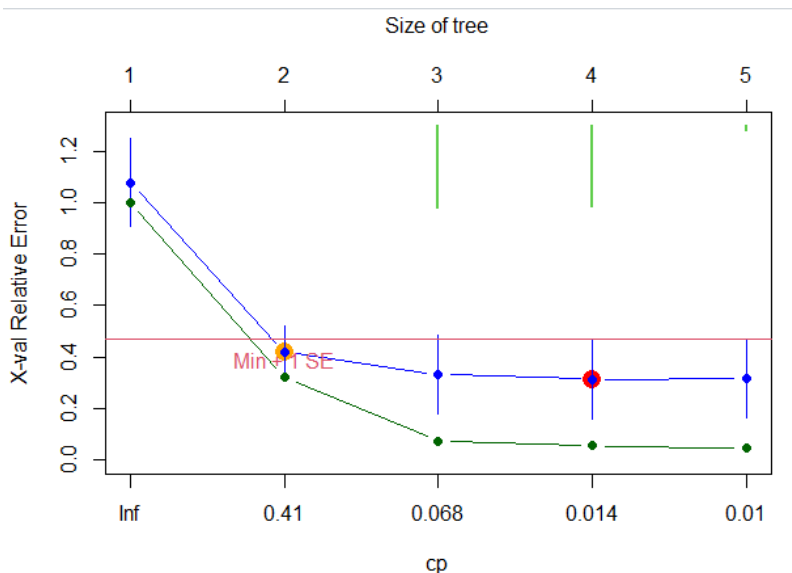
We'll specify the number of times the cross-validation process is repeated (`xvmult`) and the number of cross-validations (`xval`), and interactively choose the size of the desired tree (`xv = "pick"`):

```
Troc.terr.xv <- mvpart(spider[, "troc.terr"] ~ .,
  data = env,
  xv = "pick",
  xvmult = 100,
  control = rpart.control(xval = nrow(spider)/2),
  all.leaves = TRUE,
  plot.add = TRUE)
```

The `xval` argument sets the cross-validations to use half of the data. Note that this is specified within the `rpart.control()` function as a response to the `control` argument.

The `xvmult` argument alters the `cp` graph by adding bars showing how many of the multiple cross-validations identified a tree of that size as having the minimum cross-validated relative error. This information is also reported in the Console.

A graph of the `cp` table is displayed on the screen:



Complexity parameter table for univariate regression tree of *Trochosa terricola*, with cross-validation.

Some notes about this image:

- The green line is the relative error, as we saw previously
- The blue line is the cross-validated relative error
- The red horizontal line is one SE above the minimum cross-validated relative error
- The red point indicates the tree size that resulted in the smallest cross-validated relative error
- The orange point indicates the smallest possible tree that is below the red line (i.e., within one SE of the minimum cross-validated relative error)
- The green vertical lines descending from the top of the graph are a histogram showing how many of the cross-validations produced a tree in which the smallest cross-validated relative error was for that size. In this case, about equal numbers produced trees of size 3 or 4, and a few produced a tree of size 5.

At this point, R expects you to click on the plot to select the size of tree that you want to produce. Until you do so, the Console will be busy and will not recognize other commands. Once you have selected the size of tree that you want to produce, the rest of the function executes, and the plot of the `cp` table is replaced with the resulting tree.

Pruning a Tree

Arguments such as `minsplit` and `minbucket` provide controls on the size of the leaves that result from a regression tree. They each have defaults as described above – unless over-written, nodes are not split if they contain fewer than 5 observations (`minsplit = 5`) and every terminal node must have one-third of the observations in the preceding node (`minbucket = round(minsplit/3)`).

Pruning can be performed manually with the `snip.rpart()` function. This can be done graphically or by using the `toss` argument, which refers by number to the node(s) at which snipping is to occur.

```
snip.rpart(Troc.terr, toss = 2)
```

```
n= 28

node), split, n, deviance, yval
  * denotes terminal node

1) root 28 263.0000000 4.500000
 2) herbs< 8.5 20 79.8000000 2.900000 *
 3) herbs>=8.5 8 4.0000000 8.500000
   6) reft>=7.5 2 0.5000000 7.500000 *
   7) reft< 7.5 6 0.8333333 8.833333 *
```

By comparing this object with and without snipping, you can see that this function does not display further branching from node 2 but shows the branching from node 3. This is applied to the tree that you've already created, so the node numbers aren't changed (i.e., nodes 4 and 5 are absent in the above tree). See the help files for more information.

To snip at multiple nodes, simply include those nodes together:

```
snip.rpart(Troc.terr, toss = c(3,4))
```

```

n= 28

node), split, n, deviance, yval
  * denotes terminal node

1) root 28 263.0 4.5
  2) herbs< 8.5 20 79.8 2.9
    4) reft>=6 10 6.9 1.1 *
    5) reft< 6 10 8.1 4.7 *
  3) herbs>=8.5 8 4.0 8.5 *

```

Here, we have specified not to split nodes 3 and 4 into smaller nodes, so they are treated as terminal nodes.

Conclusions

Univariate regression trees (URT) are conceptually different than many other techniques that you've probably encountered. They can be used to evaluate a large number of explanatory variables and identify those that most strongly distinguish groups of observations with similar responses. The structure of the tree can depend strongly on which explanatory variables are included in it – particularly if they are the basis for early splits as they will determine which observations are part of each subsequent branch. Explanatory variables that are not included in the tree do not affect it.

URT is a divisive approach – it looks for ways to split a node – and is hierarchical – a split that happens early in the process constrains all subsequent decisions within the resulting nodes.

The theory behind URTs is also directly relevant for multivariate regression trees, which we consider next.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- De'ath, G. 1999. Extended dissimilarity: a method of robust estimation of ecological distances from high beta diversity data. *Plant Ecology* 144:191–199.
- De'ath, G. 2002. Multivariate regression trees: a new technique for modeling species-environment relationships. *Ecology* 83:1105–1117.
- De'ath, G., and K.E. Fabricius. 2000. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology* 81:3178–3192.
- Hothorn, T., K. Hornik, and A. Zeileis. 2006. Unbiased recursive partitioning: a conditional inference framework. *Journal of Computational and Graphical Statistics* 15:651–674.
- Vayssières, M.P., R.E. Plant, and B.H. Allen-Diaz. 2000. Classification trees: an alternative non-parametric approach for predicting species distributions. *Journal of Vegetation Science* 11:679–694.

Media Attributions

- De'ath.2002_Figure3
- URT.root
- URT.root.all.splits
- URT.split2.5
- URT.split5.5
- URT.split7.5
- URT.2split2.5
- mvpart.cp
- URT
- URT.troc.terr
- URT.troc.terr.gg
- URT.troc.terr.baseR
- URT.xv.pick

34. Multivariate Regression Trees

Learning Objectives

To understand how a multivariate regression tree (MRT) builds upon concepts related to the construction and interpretation of a univariate regression tree (URT), classifying a multivariate set of response variables into groups on the basis of one or more explanatory variables.

To consider ways to use the groups identified from a MRT.

Readings (Recommended)

De'ath (2002)

Key Packages

```
require(tidyverse, mvpart, MVPARTwrap, indicpecies)
```

Introduction

A univariate regression tree (URT) relates a single response variable to one or more explanatory variables through a series of binary splits. De'ath (2002) extended URTs to multivariate data. A multivariate regression tree (MRT) obviously requires a multivariate response.

More importantly, a MRT also requires that impurity be redefined in a multivariate sense. Three ways are available:

- **Sums of squares MRT** (SS-MRT) – “Geometrically, this is simply the sum of squared Euclidean distances of sites about the node centroid. Each split minimizes the sums of squared distances (SSD) of sites from the centroids of the nodes to which they belong. Equivalently, this maximizes the SSD between the node centroids” (De'ath 2002, p. 1107).
- **Additive MRT** – uses other sums of squares formulas, such as the sums of absolute deviations about the median.

- **Distance-based MRT** (db-MRT) – impurity is defined as the sum of within-group squared distances (as in PERMANOVA). This can be applied to any distance measure – including semimetric measures – and therefore is more general than the other ways of defining multivariate impurity. For example, it is directly applicable to community-level data. A db-MRT with Euclidean distances is exactly equivalent to a SS-MRT.

See Section 4.11 of Borcard et al. (2018) for a nice summary of MRTs.

The `mvpart()` function is built to handle both URTs and MRTs. Concepts such as pruning and cross-validation are the same with both types of data. However, not all arguments work with all methods of calculating impurity.

Each leaf/terminal node of a MRT can be characterized by:

- the multivariate mean of its observations (e.g., mean abundance of each species on the plots in that leaf/terminal node)
- the number of observations (plots)
- the environmental variables that distinguish it from others
- the species that are strongly associated with it

Of course, we're dealing with multivariate data here so the other analysis considerations that we've discussed throughout the course still apply:

- data transformation
- data standardization
- choice of distance measure (for db-MRT)

These choices can affect the results. For example, the SS-MRT reported by De'ath (2002; his Figure 2) differs from ours below because he used a different type of standardization. This is illustrated below.

Key Takeaways

A multivariate regression tree (MRT) is a direct extension of a univariate regression tree (URT). The process remains divisive and hierarchical, with a goal of making binary splits and treating each group as an independent dataset. Only two things need to change: the response needs to be multivariate, and the impurity (i.e., unexplained variance within groups) needs to be defined in a multivariate manner.

MRTs in R

We'll use the same spider data that were used to introduce URTs. However, rather than exploring the abundance of one spider species, we'll consider the composition of the spider community as a function of the measured environmental variables.

The spider abundances are already on a common scale (0-9; ordinal) so we won't relativize them. We begin by saving the data as a matrix:

```
spider.std <- data.matrix(spider[,1:12])
```

Sums of squares MRT (SS-MRT)

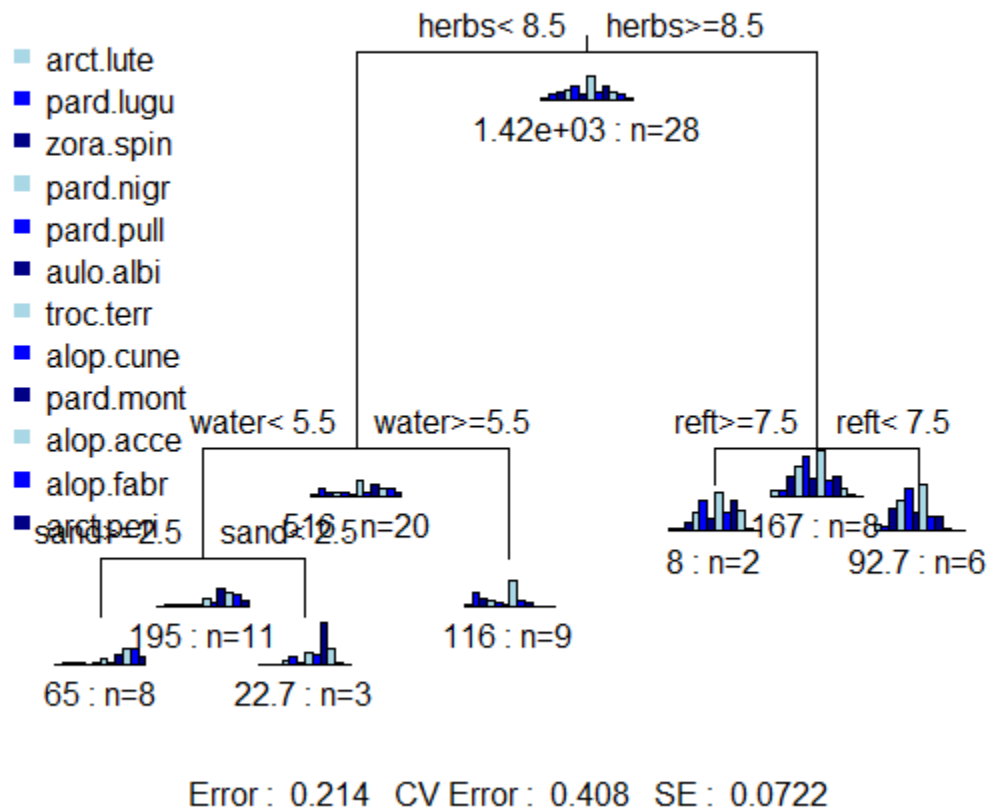
Now, we'll perform a SS-MRT on the compositional data using Euclidean distances:

```
SSMRT.spider <- mvpart(spider.std ~ .,  
  data = env,  
  xv = "pick",  
  xvmult = 100,  
  control = rpart.control(xval = nrow(spider)/2),  
  plot.add = TRUE,  
  all.leaves = TRUE,  
  dissim = "euc")
```

The **bold font** indicates the two items that changed from the last URT that we conducted (`Troc.terr.xv`):

- The left-hand side of the formula to be tested. Note that this is of class matrix; `mvpart()` automatically uses `method = "mrt"` for objects of this class. If we wanted to be more explicit about this, we could have added the `method` argument to the function. By analogy, when we considered URTs we did not specify a method and `mvpart()` automatically used `method = "anova"`.
- The argument `dissim = "euc"` was added.

We once again set `xv = "pick"` so that we could select the best tree interactively. I don't show that here, but in this case I selected a tree of size 5.



Sums of squares multivariate regression tree of the spider community as a function of six environmental variables.

The graph of a SS-MRT tree includes box plots showing the relative abundance of each species in each group. Note that these look similar to – but are definitely different from – the histograms that are sometimes provided in an URT.

Interpretation of the nodes and the explanatory variables chosen for each split are the same as for a URT.

The quantitative summary also has a very similar format to that for a URT:

```
summary(SSMRT.spider)
```

```
Call:
mvpart(form = spider.std ~ ., data = env, xv = "pick", xvmult = 100,
  plot.add = TRUE, all.leaves = TRUE, control = rpart.control(xval = nrow(spider)/2),
  dissim = "euc")
```

n= 28

```
      CP nsplit rel error
1 0.51864091      0 1.0000000
2 0.14489010      1 0.4813591
3 0.07537481      2 0.3364690
4 0.04678068      3 0.2610942
5 0.03530100      4 0.2143135
      xerror      xstd
1 1.0769716 0.12649336
2 0.5631391 0.07505089
3 0.4317718 0.07445582
4 0.4252404 0.07684907
5 0.4080949 0.07224169
```

Node number 1: 28 observations, complexity param=0.5186409
Means=0.3571,1.179,1.536,1.964,2.5,1.179,4.5,1.393,2.5,1.5,0.9286,0.4286, Summed MSE=50.0
left son=2 (20 obs) right son=3 (8 obs)

Primary splits:

```
herbs < 8.5 to the left, improve=0.5186409, (0 missing)
water < 5.5 to the left, improve=0.3015809, (0 missing)
moss < 6 to the right, improve=0.2483042, (0 missing)
reft < 7.5 to the right, improve=0.2123679, (0 missing)
sand < 5.5 to the right, improve=0.2008664, (0 missing)
```

Node number 2: 20 observations, complexity param=0.1448901
Means=0.1,1.3,0.75,0.6,0.5,0.3,2.9,0.8,2.1,1.5,1.2,0.6, Summed MSE=25.7775
left son=4 (11 obs) right son=5 (9 obs)

Primary splits:

```
water < 5.5 to the left, improve=0.3985045, (0 missing)
twigs < 3.5 to the left, improve=0.3985045, (0 missing)
reft < 3.5 to the right, improve=0.3985045, (0 missing)
moss < 6 to the right, improve=0.3518347, (0 missing)
herbs < 6.5 to the left, improve=0.2054174, (0 missing)
```

Node number 3: 8 observations, complexity param=0.04678068
Means=1,0.875,3.5,5.375,7.5,3.375,8.5,2.875,3.5,1.5,0.25,0, Summed MSE=20.875
left son=6 (2 obs) right son=7 (6 obs)

Primary splits:

```
reft < 7.5 to the right, improve=0.39720560, (0 missing)
moss < 1.5 to the right, improve=0.34610780, (0 missing)
water < 7 to the left, improve=0.20159680, (0 missing)
twigs < 1.5 to the right, improve=0.06986028, (0 missing)
```

Node number 4: 11 observations, complexity param=0.07537481
Means=0,0.1818,0.1818,0.3636,0.3636,0.1818,1.364,0.5455,3.364,2.727,2.182,1.091, Summed MSE=12.83
left son=8 (8 obs) right son=9 (3 obs)

Primary splits:

```
sand < 2.5 to the right, improve=0.54937690, (0 missing)
water < 3.5 to the left, improve=0.54454270, (0 missing)
herbs < 6.5 to the left, improve=0.43971960, (0 missing)
reft < 8.5 to the right, improve=0.20948210, (0 missing)
moss < 5.5 to the right, improve=0.09589823, (0 missing)
```

Node number 5: 9 observations
Means=0.2222,2.667,1.444,0.8889,0.6667,0.4444,4.778,1.111,0.5556,0,0,0, Summed MSE=12.83

```

Node number 6: 2 observations
Means=0.5,0.5,1.5,3.5,6,2.5,7.5,3.5,6,4,0.5,0, Summed MSE=4

Node number 7: 6 observations
Means=1.167,1,4.167,6,8,3.667,8.833,2.667,2.667,0.6667,0.1667,0, Summed MSE=15.44444

Node number 8: 8 observations
Means=0,0.25,0.25,0.25,0,0.125,1.125,0.125,1.75,2.75,2.875,1.5, Summed MSE=8.125

Node number 9: 3 observations
Means=0,0,0,0.6667,1.333,0.3333,2,1.667,7.667,2.667,0.3333,0, Summed MSE=7.555556

```

Note that the mean abundance of each species is reported for each node – these are the data that are graphed to produce the bar plots in the above graphic. The order of the species is not reported – you have to refer to the response matrix to determine which is which.

For each split, all possible breakpoints for all specified potential explanatory variables were evaluated and the variable and its breakpoint that most strongly separates the data into two groups is identified. Other variables and their optimal breakpoints are shown as well.

PS: De'ath (2002) used SS-MRT but reported a different result in his Figure 2 for the multivariate analysis of the spider community. Mike Marsh and Gavin Simpson determined that the difference is because De'ath standardized the data so that the rows and columns summed to 1. Here's code to do so:

```

spider.std1 <- scaler(spider[, 1:12], col="mean1", row="mean1")

spider.mrt1 <- mvpart(spider.std1 ~ water + sand + moss + reft + twigs + herbs, data
  = env, xv = "pick")

```

Choose a tree of size 4 to replicate his result.

Distance-based MRT (db-MRT)

We can compare the above SS-MRT with the results of a db-MRT using Euclidean distances. My testing suggests that `mvpart()` has issues with the distance matrices produced by some other functions, so we will use `gdist()`, the function provided within the `mvpart` package. In addition, we have to store the full symmetric distance matrix (not just the lower triangle); we do so via the argument `full = TRUE`.

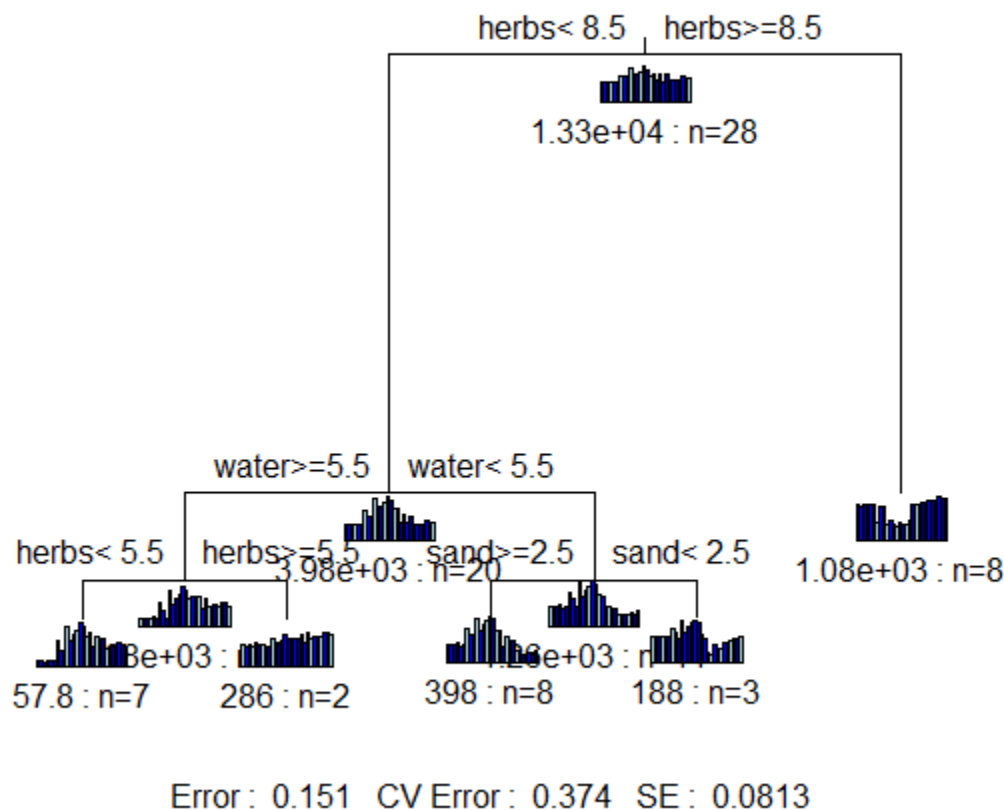
```

spider.euc <- gdist(spider.std,
  method = "euclidean",
  full = TRUE)

dbMRT.spider <- mvpart(spider.euc ~ .,
  data = env,
  xv = "pick",
  xvmult = 100,
  control = rpart.control(xval = nrow(spider)/2),
  plot.add = TRUE,
  all.leaves = TRUE)

```

For illustration purposes, I again chose a tree of size 5.



Distance-based multivariate regression tree of the spider community as a function of six environmental variables. In this case, the composition of the spider community is expressed as the Euclidean distance between sample units.

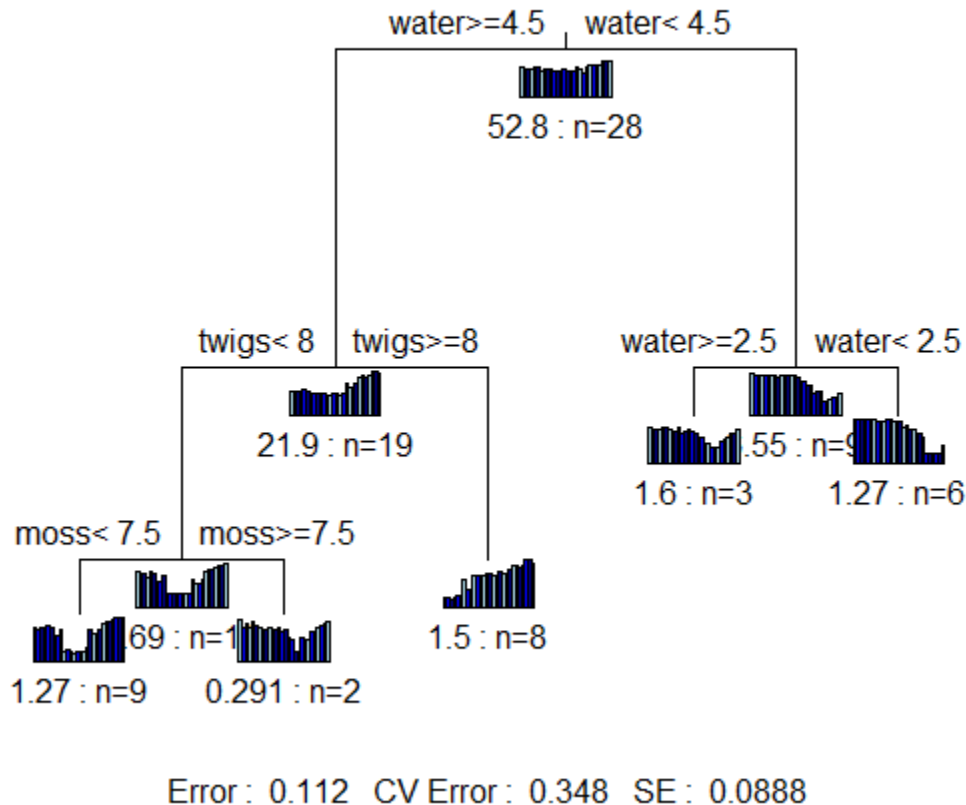
One important difference between the graphical representation of these trees: in SSMRT.spider, the bar plots are the individual species, whereas in dbMRT.spider, the bar plots are the distances between sample units (note that there are a lot more of them, and that these bar plots are not so useful).

We've talked throughout the course about how Euclidean distances are not generally appropriate for compositional data. With distance-based MRTs, we can also use other distance measures such as Bray-Curtis:

```
spider.bc <- gdist(spider.std,
  method = "bray",
  full = TRUE)

dbMRT.spider <- mvpart(spider.bc ~ .,
  data = env,
  xv = "pick", xvmult = 100,
```

```
control = rpart.control(xval = nrow(spider)/2),
plot.add = TRUE,
all.leaves = TRUE)
```



Distance-based multivariate regression tree of the spider community as a function of six environmental variables. In this case, the composition of the spider community is expressed as the Bray-Curtis distance between sample units.

Trees based on different distance measures can be directly compared. Comparing the MRTs based on Euclidean and Bray-Curtis distances:

- The MRT based on Bray-Curtis distances resulted in a slightly smaller cross-validated relative error (0.348) than that based on Euclidean distances (0.374). Other datasets may have different conclusions.
- Different variables form the basis of the splits in these trees – herbs, water, and sand for the Euclidean MRT vs. water, twigs, and moss for the Bray-Curtis MRT.
- I specified trees of the same size (5 terminal nodes) in both MRTs so I could more easily compare them, but the optimal size may differ among trees.
- Even when the trees are the same size, if they're split on the basis of different variables and breakpoints then the size of each terminal node can differ and so can the identity of the samples within each terminal node.

Interpretation of a MRT

In a MRT, we generally focus on the leaves (terminal nodes). Interpretation of a MRT can be based on various tools and techniques:

- Recognize that splits at the top of the tree are more important than those that occur near the leaves.
- Species abundances can be graphed onto the regression tree.
- The explained variance at each split for each species can be tabulated. One way to use these data is to identify species that most strongly determine the splits of the tree.
- The sample units can be ordinated (plotted in a low-dimensional space) and overlaid with information such as leaf (terminal node) identity, species abundance, etc.
- Indicator Species Analysis (ISA) can be used to identify the species that characterize each group. This can be done in several ways:
 - Two examples – the `MRT()` function and the `indicspecies` package (De Cáceres & Legendre 2009; De Cáceres et al. 2010) – are illustrated below. See here for more information about ISA.
 - Borcard et al. (2018; Section 4.12.4) discuss another way to do an ISA.
 - A slightly different approach is to relate species abundances directly to the explanatory variables – this can be done with continuously distributed variables using Threshold Indicator Taxa ANalysis (TITAN) in the `TITAN2` package (Baker & King 2010). See here for more information about TITAN.
- The resulting groups can be compared with the solution from an unconstrained cluster analysis:
 - If the unconstrained cluster analysis accounts for more of the species variation, unobserved factors (not included in the matrix of environmental variables) may be important.
 - If the two methods explain similar amounts of species variation, it is likely that important environmental variables (or their surrogates) have been identified.
 - If unconstrained clustering is weak but species – environment relationships are strong, the MRT may detect groups the unconstrained cluster analysis does not.

How to Use the Groups from a MRT

We begin by saving the group identity of each sample unit as identified in the terminal nodes of our tree. In this case, we'll use the result of the Bray-Curtis dissimilarity matrix:

```
spider$g5 <- dbMRT.spider$where
```

Alternatively, we could script the group assignments based on the regression tree. This can be helpful if we want to give the groups more informative names, for example. For each group, we simply work our way down through the tree. The `case_when()` function is very helpful here rather than using a series of `ifelse()` statements:

```
spider <- spider %>%  
  mutate(g5.alt = case_when(  
    water >= 4.5 & twigs < 8 & moss < 7.5 ~ "A",  
    water >= 4.5 & twigs < 8 & moss >= 7.5 ~ "B",  
    water >= 4.5 & twigs >= 8 ~ "C",  
    water < 4.5 & water >= 2.5 ~ "D",  
    water < 4.5 & water < 2.5 ~ "E"))
```

Note that I've simply named the groups from left to right in the tree. Comparing these two approaches:

```
table(spider$g5, spider$g5.alt)
```

```

  A B C D E
4 9 0 0 0 0
5 0 2 0 0 0
6 0 0 8 0 0
8 0 0 0 3 0
9 0 0 0 0 6

```

The two approaches assign observations to the same groups, though I named the groups with letters while the summary reported the terminal node numbers.

As with the URT, we can use this index to subset or understand our data in different ways. Two of these techniques are illustrated here. These techniques could also be used with the groups identified through a cluster analysis, etc.

Key Takeaways

Since a MRT was based on multiple responses, we can examine differences in each response variable amongst the identified groups.

Summarizing Data For Each Group

Here, we'll summarize the environmental data based on these group identities:

```

spider %>%
  select(colnames(env), g5.alt) %>%
  group_by(g5.alt) %>%
  summarize(across(everything(), mean), .groups = "keep")

```

```

# A tibble: 5 × 7
# Groups:   g5.alt [5]
  g5.alt water sand moss reft twigs herbs
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 A      7.44 0.889 3.33 5     1.44 8.67
2 B      5     0     8.5 7.5   0     7
3 C      7.75 0     1.25 0.75 9     3.12
4 D      3.67 5     6     7     0     6.67
5 E      0.5 6.83 7.83 8.5   0     4.17

```

The result is the mean value of each explanatory variable in each terminal node from the MRT. Explanatory variables that were not selected to split the tree are probably of less interest – but could be important if the chosen variables were not in the model.

We could do the same type of summary for the response variables (species abundances). See below

for an example of how Indicator Species Analysis can be used to identify the group or set of groups with which each species is most strongly associated.

Overlay Groups Onto An Ordination

We can overlay the resulting groups onto an ordination. We'll illustrate this using a NMDS ordination:

```
library(vegan)
```

```
spider.NMDS <- metaMDS(spider.bc, k = 2, wascores = FALSE)
```

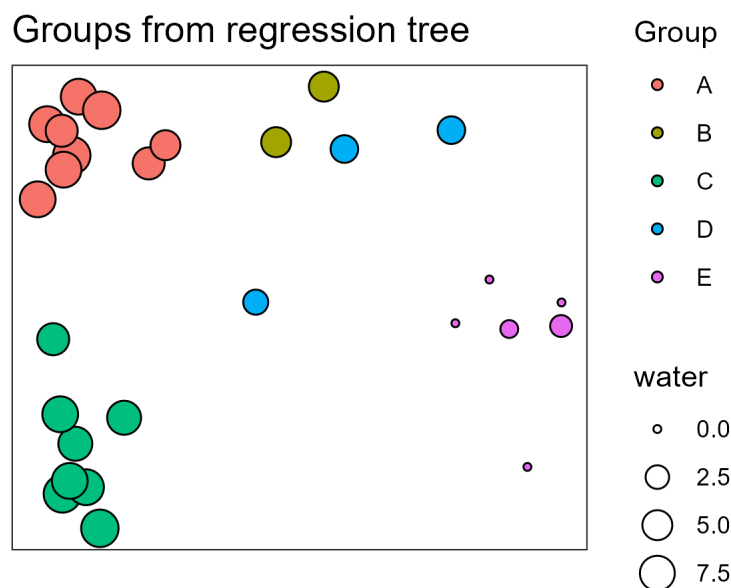
The coordinates of each plot are provided in the points aspect of this object, so let's add them to the spider object:

```
spider2 <- data.frame(spider, spider.NMDS$points)
```

We'll illustrate this using `ggplot2`, and we'll make the symbol size proportional to the value of `water` (since that was the first variable selected in the db-MRT and thus is the most important explanatory variable distinguishing the groups):

```
ggplot(data = spider2, aes(x = MDS1, y = MDS2)) +  
  geom_point(aes(fill = g5.alt, size = water), shape = 21) +  
  labs(title = "Groups from regression tree", x = "", y = "", fill = "Group") +  
  scale_x_continuous(labels = NULL, breaks = NULL) +  
  scale_y_continuous(labels = NULL, breaks = NULL) +  
  theme_bw()
```

```
ggsave("graphics/spider_tree.png", width = 4, height = 3, units = "in", dpi = 300)
```



NMDS ordination of the spider community, with sample units color coded based on the group they were assigned to through a distance-based MRT. Symbol size is proportional to the abundance of water in the sample units as water was the basis for the first binary split of the data.

We'll discuss ordinations soon, but for now I'll note that this ordination is based entirely on the distance matrix derived from the response variables (species abundances). In contrast, the tree and associated classification are based on the relationship between that same distance matrix and the explanatory variables. Alignment between the ordination and the regression tree – observations that are close to one another in the ordination also being assigned to the same group in the MRT – is an indication that composition varies substantially with respect to **water**.

In this graphic, point size relates to **water**, which was the basis for the first split in this tree and point color relates to the groups identified in the MRT. Group E, those associated with the lowest values for water, are on one end of the ordination while groups with higher values for water are on the other end of the ordination.

The regression tree (shown above) indicates that **twigs** and **moss** are also important in distinguishing some groups, though we have not incorporated them into this image.

Explore Other Aspects of MRT Output

The **MVPARTwrap** package is not actively maintained and therefore has to be installed from github:

```
devtools::install_github("cran/MVPARTwrap", force = TRUE)
```

```
library(MVPARTwrap)
```

It includes the **MRT()** function, which is a wrapper that provides additional output from a MRT analysis. Its usage is:

```
MRT(
  obj,
  percent,
  species = NULL,
  LABELS = FALSE,
  ...
)
```

The key arguments are:

- **obj** – the object to be summarized (class **mvpart**).
- **percent** – the minimum importance of a species. Summary will highlight species that account for at least this much variance at a node. Default not specified, but appears to be 10.
- **species** – a vector of species names.

The output includes:

- **nodes** – node numbers, in increasing order of contribution to explained variance.
- **pourct** – species contributions to explained variation (%) at each node. Each row is a node, and each column is a species. Note that this order is transposed from that in **TABLE1** below. In addition, the values here are percentages of the column totals from **TABLE1**.
- **R2** – species contribution to explained variation at each node. R^2 is equal to 1 minus the relative error. Same information for each species as in **TABLE1**, but expressed as a proportion and in transposed order.
- **RWHERE**, **LWHERE** – a list of vectors containing the row numbers of objects or left or right side of each node.
- **TABLE1** – matrix showing how total species variation is partitioned – see example below. Each

species is a row, plus there is a final row (total_col) that is the sum of that column. There are multiple columns:

- One column per node
- col_total_tree – percent of variation within tree that is explained by that species. The value in the 'total_col' row is the total R2 of the tree.
- col_total_species – percent of variation in entire data matrix (total_col = 100%) and the proportion of that variation associated with each species.

We can apply this to our SSMRT example and index desired elements:

```
MRT(SSMRT.spider)$TABLE1 %>% round(1)
```

	herbs8.5	water5.5	sand2.5	reft7.5
arct.lute	0.3	0.0	0.0	0.0
pard.lugu	0.1	2.2	0.0	0.0
zora.spin	3.0	0.6	0.0	0.8
pard.nigr	9.2	0.1	0.0	0.7
pard.pull	19.7	0.0	0.3	0.4
aulo.albi	3.8	0.0	0.0	0.1
troc.terr	12.6	4.1	0.1	0.2
alop.cune	1.7	0.1	0.4	0.1
pard.mont	0.8	2.8	5.4	1.2
alop.acce	0.0	2.6	0.0	1.2
alop.fabr	0.4	1.7	1.0	0.0
arct.peri	0.1	0.4	0.3	0.0
total_col	51.9	14.5	7.5	4.7

	col_total_tree	col_total_species
arct.lute	0.4	1.0
pard.lugu	2.3	4.7
zora.spin	4.4	6.0
pard.nigr	10.0	14.5
pard.pull	20.5	21.9
aulo.albi	4.0	4.9
troc.terr	17.0	18.5
alop.cune	2.3	4.4
pard.mont	10.1	13.3
alop.acce	3.8	5.3
alop.fabr	3.0	3.7
arct.peri	0.9	1.8
total_col	78.6	100.0

The values here relate to the relative importance of each species (row) with respect to each split (column). For example, the first split (herbs8.5) is primarily associated with variation in two species, pard.pull and troc.terr. The values in the 'total_col' row are (100 * cp) associated with each node – for example, 0.519 for the first split (confirm above).

Indicator Species Analysis of Groups

We'll talk about Indicator Species Analysis (ISA) later, but here's an example of its application to the groups identified from a MRT. It can be equivalently applied to the product of a hierarchical cluster analysis, *k*-means cluster analysis, etc.

```
library(indicspecies)

summary(multipatt(spider[,1:12],
  cluster = spider$g5.alt))
```

Multilevel pattern analysis

Association function: IndVal.g
Significance level (alpha): 0.05

Total number of species: 12
Selected number of species: 12
Number of species associated to 1 group: 2
Number of species associated to 2 groups: 5
Number of species associated to 3 groups: 2
Number of species associated to 4 groups: 3

List of species associated to each combination:

Group A #sps. 1
stat p.value
arct.lute 0.882 0.015 *

Group E #sps. 1
stat p.value
arct.peri 1 0.005 **

Group A+B #sps. 2
stat p.value
pard.pull 0.968 0.005 **
pard.nigr 0.927 0.005 **

Group A+C #sps. 1
stat p.value
pard.lugu 0.883 0.01 **

Group A+D #sps. 1
stat p.value
aulo.albi 0.928 0.005 **

Group D+E #sps. 1
stat p.value
alop.fabr 0.978 0.005 **

Group A+C+D #sps. 1
stat p.value
zora.spin 0.922 0.01 **

Group B+D+E #sps. 1
stat p.value
alop.acce 0.924 0.005 **

Group A+B+C+D #sps. 2
stat p.value
troc.terr 0.981 0.015 *
alop.cune 0.929 0.005 **

```

Group A+B+D+E #sps. 1
      stat p.value
pard.mont 0.968 0.005 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The test statistic here (`stat`) ranges from 0 to 1; the larger the value the more strongly that species is associated with that group.

Some species are associated with individual groups. For example, if you sampled a new location and it contained `arct.lute`, that location would likely belong to group A. On the other hand, if it contained `arct.peri`, the location would likely belong to group E.

Other species are associated with two or more groups. When a species is strongly associated with multiple groups, it can be helpful to think instead of the groups where it does not occur. For example, the spider `pard.mont` is an indicator of all groups except group C, whereas `alop.cune` is an indicator of all groups except group E.

Another way to use this is to consider which species are indicators of a particular group, either individually or as part of a set of groups. For example, if a sample was from group D, the above results suggest that it would likely contain:

- `aulo.albi`
- `alop.fabr`
- `zora.spin`
- `alop.acce`
- `troc.terr`
- `alop.cune`
- `pard.mont`

The results of an ISA can also be linked back to the explanatory variables selected in the MRT.

Conclusions

It seems to me that multivariate regression trees are underutilized in ecology at present. There are many creative ways they could be used, including as a way to identify spatial or temporal discontinuities in composition (Borcard et al. 2018). However, the `mvpart` package or a comparable package would have to be updated and maintained for this to be feasible.

Distance-based MRTs are particularly appealing because they can be applied to any distance matrix regardless of the choice of distance measure.

References

Baker, M.E., and R.S. King. 2010. A new method for detecting and interpreting biodiversity and ecological community thresholds. *Methods in Ecology and Evolution* 1:25-37.

Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.

De'ath, G. 2002. Multivariate regression trees: a new technique for modeling species-environment relationships. *Ecology* 83:1105-1117.

De Cáceres, M., and P. Legendre. 2009. Associations between species and groups of sites: indices and statistical inference. *Ecology* 90:3566-3574.

De Cáceres, M., P. Legendre, and M. Moretti. 2010. Improving indicator species analysis by combining groups of sites. *Oikos* 119:1674-1684.

Media Attributions

- MRT.spider.SSMRT
- MRT.spider.dbMRT
- MRT.spider.dbMRT.bc
- MRT.spider.NMDS

PART IV

ORDINATIONS (DATA REDUCTION AND VISUALIZATION)

Analyses of *a priori* hypotheses about multivariate data may be summarized as a series of steps (modified from Anderson 2001):

1. Choose transformation and standardization (if any) to apply to the data
2. Choose appropriate distance measure
3. Conduct statistical tests
4. Visualize patterns of resemblance among observations

Similarly, classification of sample units into groups is a series of steps:

1. Choose transformation and standardization (if any) to apply to the data
2. Choose appropriate distance measure
3. Apply clustering or other algorithm to identify groups
4. Visualize patterns of resemblance among observations

Step 3 is the only one that differs between these two lists. We've covered steps 1, 2, and 3 already. **Visualizing patterns** (step 4) simply means illustrating what the statistical tests have identified or showing the groups that have been identified through a classification process. This is one of the key purposes of ordination. Visualization is common but optional; there is no statistical test associated with an ordination.

Another purpose of ordinating data is to **reduce data dimensionality**. Ordinations seek to reduce the dimensionality of a dataset while minimizing the amount of information that is lost. This is necessary to visualize patterns but can also be useful in other ways. In some instances, for example, the reduced data themselves subjected to statistical analysis.

In this section, we will consider several types of ordination methods:

- Principal Component Analysis (PCA)
- Correspondence Analysis (CCA) and variants (DCA, CCA, DCCA)
- Non-metric MultiDimensional Scaling (NMDS),
- Principal Coordinates Analysis (PCoA; aka metric multidimensional scaling (MDS)),
- Redundancy Analysis (RDA)
- Distance-based Redundancy Analysis (db-RDA; aka Canonical Analysis of Principal Coordinates (CAP))

After surveying a range of ordinations, we will compare ordination techniques, review general graphing principles, and focus on how to visualize and interpret ordinations.

References

Anderson, M.J. 2001. A new method for non-parametric multivariate analysis of variance. *Austral Ecology* 26:32-46.

35. Types of Ordination Methods

Learning Objectives

To summarize the characteristics and types of ordination techniques available.

Introduction

An ordination is a “graphical representation of the similarity of sampling units and/or attributes in resemblance space” (Wildi 2010, p. 35). The ‘attributes’ in this quote are the suite of response variables – species, traits, chemical concentrations, etc. The ‘resemblance space’ is the distance or dissimilarity measure used to synthesize the attributes on each sampling unit.

Ordination techniques summarize the data in a reduced number of dimensions while accounting for as much of the variability in the original data set as possible. As a result, they enable you to visualize relationships among sample units or with respect to individual variables.

Characteristics of Ordination

One author (<http://ordination.okstate.edu/ideal.htm>) suggests the following desirable qualities for an ordination:

- It recovers gradients without distortion.
- If clusters exist in nature, they should be obvious in the ordination.
- It does not produce clusters which do not exist in nature.
- It gives the same result every time for a given data set.
- There is a unique solution.
- Ecological similarity is related to proximity in ordination space.
- Scaling of axes is related to beta diversity.
- Not sensitive to noise.
- “Signal” and “Noise” are easily separated.
- You do not need to pre-specify the number of axes.
- The solution is the same, no matter how many dimensions one chooses to look at.
- Unless by choice, all sample units are treated equally.
- The solution does not take much computer time.
- The method is robust – it works well for:
 - short and long gradients
 - low and high noise
 - sparse and full matrices
 - big and small data sets
 - species-rich and species-poor systems
- For the mathematician: elegant.
- For the ecologist: available, inexpensive, and easy to understand.

The items on this 'wish list' are not equal, and items that are important to one person may not be important to another. For example, we have enough computing power these days that computer time is no longer an issue for most of these techniques.

There is no "ideal" ordination that possesses all of these qualities. Different techniques emphasize different qualities.

Types of Ordination

The following distinctions (from <http://ordination.okstate.edu/>) may be helpful in categorizing different types of ordinations.

Indirect Gradient Analysis

Indirect gradient analysis utilizes a single matrix containing the response variables (e.g., sample units x species). These methods are also described as being **unconstrained** because they do not consider other data (environmental characteristics, grouping variables, spatial location, etc.). Those other data may, however, be used in a post-hoc fashion to interpret the results of an indirect gradient analysis.

Indirect gradient analysis can be conducted via **eigenanalysis** or on the basis of a **distance matrix**.

Eigenanalysis-based approaches produce a set of **eigenvalues**, each of which has an associated **eigenvectors**. These approaches are further distinguished by whether they assume **linear** or **unimodal** relationships among variables:

- Principal Component Analysis (PCA) assumes a linear relationships among variables
- Correspondence Analysis (CA) and Detrended Correspondence Analysis (DCA) assume a unimodal relationship among variables

Distance-based approaches do not make assumptions about how the variables relate to one another, and instead focus on the patterns contained within the distance matrix. I use 'distance' here as a general term that encompasses both strict distance matrices (produced by metric measures) and dissimilarity matrices (produced by semi-metric measures) – see the chapter about distance measures for a refresher on this distinction. Examples include:

- Non-metric MultiDimensional Scaling (NMDS)
- Principal Coordinates Analysis (PCoA; aka metric multidimensional scaling (MDS))
- Polar ordination (aka Bray-Curtis ordination)

Direct Gradient Analysis

Direct gradient analysis methods relate a matrix of response variables to one or more explanatory variables. As a result, this type of analysis is **constrained**. Given this focus, it should be apparent that these analyses require two objects – a matrix of response variables (e.g., sample units x species) and a matrix or dataframe containing one or more explanatory variables associated with each sample unit. These types of ordinations are conceptually similar to regression, where the objective is to relate sample units on the basis of their values in the two matrices.

Direct gradient analysis can be conducted assuming **linear** or **unimodal** relationships among variables:

- ReDundancy Analysis (RDA) assumes a linear relationship among variables
- Canonical Correspondence Analysis (CCA) and Detrended Canonical Correspondence Analysis (DCCA) assume a unimodal relationship among variables
- Distance-based Redundancy Analysis (db-RDA; aka Canonical Analysis of Principal Coordinates (CAP)) is analogous to RDA but, as the name suggests, focuses on the distance matrix

References

Wildi, O. 2010. *Data analysis in vegetation ecology*. Wiley-Blackwell, West Sussex, UK.

36. PCA

Learning Objectives

To consider how correlated variables can be combined into uncorrelated principal components.

To demonstrate how to use PCA to rotate and translate data, and to reduce data dimensionality.

To explain how the eigenvalue and eigenvector of a principal component relate to its importance and loadings, respectively.

To introduce the biplot, a common technique for visualizing the results of a PCA.

Key Packages

```
require(stats, vegan, tidyverse, ggbiplot)
```

Introduction

Principal Component Analysis (PCA) is an eigenanalysis-based approach. We begin, therefore, by briefly reviewing eigenanalysis. For more details on this topic, refer to the chapter about Matrix Algebra.

Review of Eigenanalysis

Eigenanalysis is a method of identifying a set of linear equations that summarize a symmetric square matrix. It yields a set of **eigenvalues** (λ), each of which has an associated **eigenvector** (\mathbf{x}). The connection between these terms is expressed in Equation A.16 from Gotelli & Ellison (2004):

$$\mathbf{Ax} = \lambda\mathbf{x}$$

In words, this says that a square matrix \mathbf{A} can be multiplied by a vector \mathbf{x} and yield the same values as a scalar value λ multiplied by the same vector \mathbf{x} . While this may not sound very helpful, it is **the basis for ordination techniques such as PCA**. It means that data can be rotated, reflected, stretched, or compressed in coordinate-space by multiplying the individual data points by an eigenvector (\mathbf{x}). Each combination of an eigenvalue (λ) and its associated eigenvector (\mathbf{x}) is a solution for Gotelli & Ellison's Equation A.16.

The **eigenvalues** (λ in Equation A.16) represent the variance extracted by each axis. There are as many eigenvalues as there are rows or columns in the matrix (it doesn't matter which you consider, as the matrix is symmetric), and the eigenvalues are sorted in order of decreasing size. In other words, the first axis always accounts for the most variance, the second axis for the next most variance, etc. They are usually reported as a percentage of the total variance.

The **eigenvectors** (\mathbf{x} in Equation A.16) are associated with the eigenvalue in the same relative position – i.e., the first eigenvector (first column) is associated with the first eigenvalue. The sign of the eigenvector may vary among analyses. For example, an element may be positive in one analysis but negative in another. These analyses are equivalent; they are just mirror images of one another.

Another way to describe eigenanalysis is the process of identifying and computing a new coordinate system for the data. We conventionally represent data graphically such that each axis is a response variable. Eigenanalysis allows us to create new axes that are combinations of the response variables.

An important benefit of eigenanalysis is that the **axes of the new coordinate system are uncorrelated with one another**. The weights (eigenvectors) are chosen so that the value for a sample unit on one axis is unrelated to the value for that same sample unit on another axis.

Theory

PCA was developed by Karl Pearson (1901), who used it to determine racial assignment of individuals based on multiple biometric measurements (Gotelli & Ellison 2004). Harold Hotelling (1933) developed the mathematics behind PCA, and Goodall (1954) introduced it to the ecological literature under the term 'factor analysis' (which is now used to indicate a different analytic technique).

It's easiest to use an example to illustrate how PCA works. Imagine your data as a cloud of data points. PCA begins by identifying the best-fitting straight line through the cloud. This line is oriented so that it explains as much of the variability in the data cloud as possible, and is called the first **principal component (PC)**. The location of each sample along this axis is calculated by combining its values for the original variables.

Now, imagine rotating the data cloud around the first principal axis until you can identify a second axis that explains as much of the remaining variation as possible. Since you are holding the first axis fixed, the second axis will be perpendicular to (orthogonal to) it. Again, the location of each sample along this axis is calculated by combining its values for the original variables.

Repeat this process for every subsequent axis: rotate the data cloud around the axes that have already been identified and determine the axis that explains as much of the remaining variation as possible while remaining orthogonal to all other axes. Note that although we can perform this mathematically for any number of dimensions, we cannot visualize it in more than three dimensions!

In brief, PCA identifies a series of new 'synthetic' variables (**principal components; PCs**) that are composites or blends of the original variables.

- Each eigenvalue measures the amount of variance explained by that principal component.
- The first eigenvalue accounts for as much variation as possible, the second as much of the remaining variation as possible, etc.
- Each eigenvector is the set of weights assigned to the original variables to produce that principal component.
- Each principal component is uncorrelated with the others.

The effectiveness of PCA is a function of how strongly correlated the original variables are:

- If the variables are highly correlated, PCA can very effectively express them in a smaller number of principal components.
- If all but one variable are highly correlated, one PC will distinguish the uncorrelated variable from the others and the rest of the PCs will largely be combinations of the correlated variables.
- If the variables are uncorrelated, there by definition is no 'shared' information between them and each principal component is largely associated with one of the variables.

We can use a PCA in several ways:

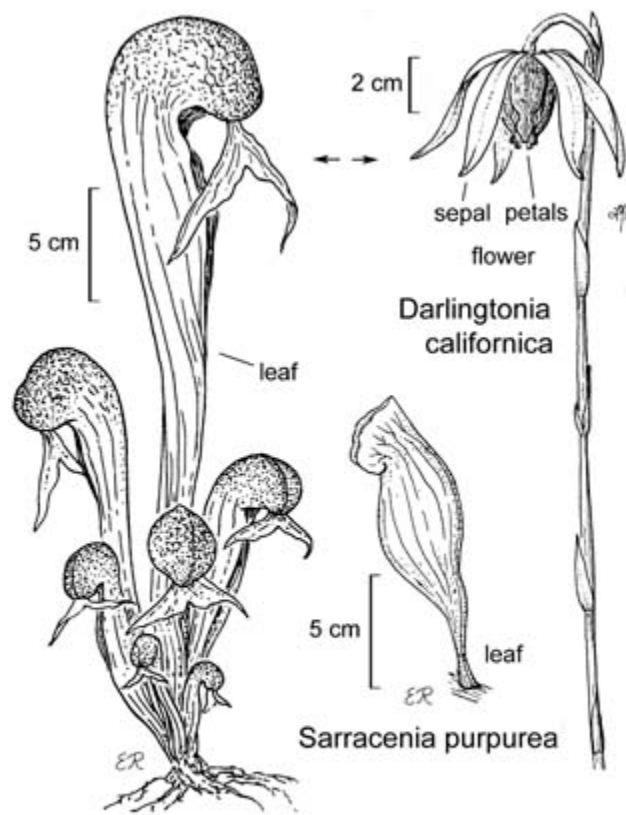
- If we keep all of the PCs, we have **rotated** the data without altering the distances among observations.
- If we focus on the first few PCs, we have **reduced the dimensionality** of the dataset.
- Since the eigenvalues are in declining order of importance, focusing on them in order means that we are focusing on as much variation as possible to explain in that number of dimensions. For example, it is impossible to explain more variation in one dimension than is explained by PC1, to explain more variation in two dimensions than is explained by PC1 + PC2, etc.
- The principal components are uncorrelated with (**orthogonal** to) each other. Therefore, the individual components may better meet the assumptions of other analytical techniques (multiple regression, MANOVA, etc.) than the original variables could, and can be analyzed with those techniques.
- Since each principal component is a blend of the original variables, it synthesizes all of those variables simultaneously. It can be more robust to analyze the principal component rather than focusing on one variable. If you chose to analyze one variable, you would have to justify its choice. And, your conclusions might depend on which variable you chose.

PCA is a very important technique to understand. More details about PCA can be found in Borcard et al. (2018; Section 5.3) and Legendre & Legendre (2012; Section 9.1).

PCA is appropriate for many types of data (e.g., LiDAR, morphological data). Summerville et al. (2006) apply this technique to identify suites of correlated traits among moth species, and then analyze each suite of traits (principal component) individually. However, PCA is widely acknowledged to be inappropriate for community-level analyses that involve sample × species data matrices (see 'Conclusions' section for details).

***Darlingtonia* Dataset**

We'll illustrate the steps of a PCA using a dataset containing morphological measurements of *Darlingtonia californica* (cobra lily, or California pitcherplant) plants. *Darlingtonia* are carnivorous plants with specialized leaves that function as pitchers which trap insects.



Darlingtonia californica

Sarraceniaceae

© Regents of the University of California

Line drawing of *Darlingtonia californica* (California pitcherplant).

This dataset is taken from Table 12.1 of Gotelli & Ellison (2004). Measurements were made on 87 plants from four sites in southern Oregon. Ten variables were measured on each plant, 7 based on morphology and 3 based on biomass:

Name	Type	Description	Units
height	Morphological	Plant height	mm
mouth.diam	Morphological	Diameter of the opening (mouth) of the tube	mm
tube.diam	Morphological	Diameter of the tube	mm
keel.diam	Morphological	Diameter of the keel	mm
wing1.length	Morphological	Length of one of the two wings in the fishtail appendage (in front of the opening to the tube)	mm
wing2.length	Morphological	Length of second of the two wings in the fishtail appendage (in front of the opening to the tube)	mm
wingsprea	Morphological	Length from tip of one wing to tip of second wing	mm
hoodmass.g	Biomass	Dry weight of hood	g
tubemass.g	Biomass	Dry weight of tube	g
wingmass.g	Biomass	Dry weight of fishtail appendage	g

The datafile is available through the GitHub repository. Save it to the 'data' sub-folder within the folder containing your SEFS 502 project.

Open your course R project and import the data:

```
darl <- read.csv("data/Darlingtonia_GE_Table12.1.csv",
  header = TRUE)
```

Create an object that contains the response variables (i.e., omitting the columns that code for site and plant number):

```
darl.data <- darl %>%
  select(-c(site, plant))
```

The Steps of a PCA

Our data are often summarized in a data matrix (n sample units \times p response variables). Most of the techniques we've considered focus on the $n \times n$ distance matrix. With PCA, we focus instead on the p response variables. We will express them as a $p \times p$ matrix, but note that this is not a distance matrix – see Step 2 below for details.

We can think of a PCA as a series of 6 steps. The first two steps can be done in two ways (options A and B).

Step 1: Normalize Data (optional)

Option A: Normalize data – subtract the mean and divide by the standard deviation for each column. This expresses all variables on the same units: how many standard deviations each

observation is above or below the mean for a variable. The centroid of the normalized dataset is therefore at the origin.

```
scaled.data <- scale(dar1.data)
```

If variables are not on the same scale, they will contribute unequally to the results: a variable with high variance will contribute much more than a variable with low variance.

Option B: Keep data in raw format.

Step 2: Calculate Variance/Covariance Matrix or Correlation Matrix

Option A: Calculate the variance/covariance matrix (**S**) based on the scaled data:

```
S <- cov(scaled.data)
```

```
S %>% round(2)
```

	height	mouth.diam	tube.diam	keel.diam	wing1.length	
height	1.00	0.61	0.24	-0.03	0.28	
mouth.diam	0.61	1.00	-0.05	-0.35	0.57	
tube.diam	0.24	-0.05	1.00	0.54	-0.09	
keel.diam	-0.03	-0.35	0.54	1.00	-0.33	
wing1.length	0.28	0.57	-0.09	-0.33	1.00	
wing2.length	0.29	0.44	0.01	-0.26	0.84	
wingsprea	0.12	0.25	0.10	-0.21	0.64	
hoodmass.g	0.49	0.76	-0.11	-0.30	0.61	
tubemass.g	0.84	0.72	0.10	-0.25	0.44	
wingmass.g	0.10	0.24	0.03	-0.12	0.47	
	wing2.length	wingsprea	hoodmass.g	tubemass.g	wingmass.g	
height	0.29	0.12	0.49	0.84	0.10	
mouth.diam	0.44	0.25	0.76	0.72	0.24	
tube.diam	0.01	0.10	-0.11	0.10	0.03	
keel.diam	-0.26	-0.21	-0.30	-0.25	-0.12	
wing1.length	0.84	0.64	0.61	0.44	0.47	
wing2.length	1.00	0.71	0.49	0.36	0.47	
wingsprea	0.71	1.00	0.25	0.16	0.34	
hoodmass.g	0.49	0.25	1.00	0.76	0.35	
tubemass.g	0.36	0.16	0.76	1.00	0.17	
wingmass.g	0.47	0.34	0.35	0.17	1.00	

Option B: Convert raw data to correlation matrix (**P**).

```
P <- cor(dar1.data)
```

```
P %>% round(2)
```

	height	mouth.diam	tube.diam	keel.diam	wing1.length	
height	1.00	0.61	0.24	-0.03	0.28	
mouth.diam	0.61	1.00	-0.05	-0.35	0.57	
tube.diam	0.24	-0.05	1.00	0.54	-0.09	
keel.diam	-0.03	-0.35	0.54	1.00	-0.33	
wing1.length	0.28	0.57	-0.09	-0.33	1.00	
wing2.length	0.29	0.44	0.01	-0.26	0.84	
wingsprea	0.12	0.25	0.10	-0.21	0.64	
hoodmass.g	0.49	0.76	-0.11	-0.30	0.61	
tubemass.g	0.84	0.72	0.10	-0.25	0.44	
wingmass.g	0.10	0.24	0.03	-0.12	0.47	
	wing2.length	wingsprea	hoodmass.g	tubemass.g	wingmass.g	
height	0.29	0.12	0.49	0.84	0.10	
mouth.diam	0.44	0.25	0.76	0.72	0.24	
tube.diam	0.01	0.10	-0.11	0.10	0.03	
keel.diam	-0.26	-0.21	-0.30	-0.25	-0.12	
wing1.length	0.84	0.64	0.61	0.44	0.47	
wing2.length	1.00	0.71	0.49	0.36	0.47	
wingsprea	0.71	1.00	0.25	0.16	0.34	
hoodmass.g	0.49	0.25	1.00	0.76	0.35	
tubemass.g	0.36	0.16	0.76	1.00	0.17	
wingmass.g	0.47	0.34	0.35	0.17	1.00	

Look at **S** and **P** – they’re identical! This means that we can work with either one in all subsequent steps.

(Note: this is only true if the variables have been normalized in Option A. If you centered but did not rescale them, each variable would be weighted in proportion to its variance. In my experience this is uncommon.)

Step 3: Conduct Eigenanalysis

Conduct eigenanalysis of **S** (or **P**; the results will be identical as long as **S == P**).

```
S.eigen <- eigen(S)
```

Step 4: Interpret the Eigenvalues

There will be as many eigenvalues as there are variables.

```
S.eigen$values %>% round(3) # Eigenvalues
```

```
[1] 4.439 1.768 1.531 0.734 0.473 0.362 0.253 0.244 0.143 0.053
```

The eigenvalues sum to the total of the diagonal of the matrix analyzed. If the data have been normalized (**S**) or are based on a correlation matrix (**P**), the diagonal values are all 1 (see above) and hence the eigenvalues sum to the number of variables:

```
sum(S.eigen$values)
```

```
[1] 10
```

Eigenvalues are generally expressed as a proportion of the total amount of variation:

```
S.eigen.prop <- S.eigen$values / sum(S.eigen$values)
```

```
S.eigen.prop %>% round(3) # rounding for display
```

```
[1] 0.444 0.177 0.153 0.073 0.047 0.036 0.025 0.024 0.014 0.005
```

The first eigenvalue always accounts for the most variance, and the proportion of variation declines for each subsequent eigenvalue. The cumulative variance explained is analogous to the R^2 value from a regression. Each eigenvalue is associated with a principal component.

One of the important decisions to be made here is how many eigenvalues to retain; see 'How Many Principal Components?' section below for details. In this example, we will focus on the first three principal components. Together, these three PCs account for $0.444 + 0.177 + 0.153 = 0.774$, or 77% of the variance in the 10 response variables.

Step 5: Interpret the Eigenvectors

Every eigenvalue has an associated eigenvector:

```
S.eigen$vectors %>% round(2) # 1 vector per eigenvalue
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] -0.30 -0.49 -0.03  0.12  0.29  0.53  0.06  0.19  0.03 -0.50
[2,] -0.39 -0.20  0.18  0.00 -0.14 -0.42  0.24  0.69  0.13  0.19
[3,]  0.02 -0.35 -0.63  0.07  0.28 -0.54  0.20 -0.24 -0.01 -0.09
[4,]  0.20 -0.29 -0.53 -0.16 -0.63  0.28 -0.16  0.22 -0.03  0.16
[5,] -0.40  0.26 -0.11  0.07 -0.30  0.03  0.37 -0.12 -0.70 -0.17
[6,] -0.37  0.28 -0.24  0.17 -0.14  0.24  0.36 -0.23  0.64  0.16
[7,] -0.27  0.35 -0.34  0.44  0.17 -0.03 -0.61  0.28 -0.07  0.02
[8,] -0.40 -0.13  0.16 -0.23 -0.39 -0.29 -0.44 -0.33  0.20 -0.40
[9,] -0.37 -0.40  0.12  0.02  0.13  0.13 -0.19 -0.33 -0.21  0.68
[10,] -0.23  0.25 -0.24 -0.82  0.34  0.10 -0.05  0.13 -0.02  0.05
```

In this matrix, the rows correspond to the original variables, in order, and the columns correspond to the principal component identified by each eigenvalue.

The elements of an eigenvector are known as the **loadings**. The loadings are the coefficients of the linear equation that 'blend' the variables together for that principal component. Interpretation is based on these loadings. Since the first three PCs account for most of the variation in this dataset, let's focus on them. We'll add row and column names to aid in interpretation.

```
loadings <- S.eigen$vectors[ , 1:3] %>%  
  data.frame(row.names = colnames(dar1.data)) %>%  
  rename("PC1" = X1, "PC2" = X2, "PC3" = X3) %>%  
  round(digits = 3)
```

loadings

	PC1	PC2	PC3
height	-0.297	-0.494	-0.026
mouth.diam	-0.390	-0.199	0.178
tube.diam	0.016	-0.346	-0.635
keel.diam	0.197	-0.290	-0.527
wing1.length	-0.399	0.257	-0.114
wing2.length	-0.371	0.284	-0.243
wingsprea	-0.272	0.354	-0.344
hoodmass.g	-0.396	-0.133	0.158
tubemass.g	-0.371	-0.400	0.124
wingmass.g	-0.234	0.249	-0.238

When summarizing a PCA in tabular form, two additional elements are often added as additional rows below the loadings:

- Proportion of variance explained by each PC
- Cumulative proportion of variance explained by this and larger PCs

The proportions of variance explained are helpful to remind us of the relative importance of the PCs, though these proportions obviously should not be interpreted like loadings!

Interpretation of the loadings is subjective, but is based on their magnitude (absolute value) and direction. Summerville et al. (2006) restricted their attention to loadings $> |0.4|$ but you can find examples of other thresholds in the literature.

A variable is most important for the principal component where its loading has the largest absolute value. For example, review the above table of loadings and confirm that height is more strongly related to PC2 than to the other PCs and that mouth.diam is more strongly related to PC1 than to the other PCs.

The set of loadings that comprise a PC should also be compared:

- Larger magnitude = more weight
- Similar magnitude and same direction = similar weight (both variables increase at same time)
- Similar magnitude but opposite direction = contrasting weight (one variable increases as the other decreases)

After examining the magnitude and direction of the loadings, here is how we might interpret the results of this PCA:

- PC1 gives roughly equal weight to all variables except tube.diam and keel.diam, and can be interpreted as a measure of pitcher size. This PC accounts for 44% of the variance in the response variables.
- PC2 gives opposing weights to pitcher height and diameter than to the dimensions of the fishtail appendage (wing1.length, wing2.length, wingsprea), and thus can be interpreted as a measure of pitcher shape. This PC accounts for 18% of the variance in the response variables.
- PC3 gives most weight to tube and keel diameters, and might be interpreted as a measure of leaf width. This PC accounts for 15% of the variance in the response variables.

Step 6: Interpret the Scores of Sample Units

The score or location of each sample unit on each axis is determined by matrix-multiplying the data matrix by the eigenvector for that axis.

For example, the scores for all sample units for the first **principal component** are:

```
PC1 <- scaled.data %*% S.eigen$vectors[,1]
PC1 %>% head()
```

```
      [,1]
[1,] -1.4648658
[2,]  3.2696887
[3,]  1.0026952
[4,] -0.7149071
[5,]  0.9887680
[6,] -1.2136554
```

The scores for all 87 plants for all PCs are calculated in the same way:

```
PC.scores <- scaled.data %*% S.eigen$vectors
```

I haven't displayed PC.scores here as it is large (87 rows x 10 columns ... right?).

The PC scores are orthogonal to (i.e., uncorrelated with) each other:

```
cor(PC.scores) %>% round(3)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    0    0    0    0    0    0    0    0    0
```

[2,]	0	1	0	0	0	0	0	0	0	0
[3,]	0	0	1	0	0	0	0	0	0	0
[4,]	0	0	0	1	0	0	0	0	0	0
[5,]	0	0	0	0	1	0	0	0	0	0
[6,]	0	0	0	0	0	1	0	0	0	0
[7,]	0	0	0	0	0	0	1	0	0	0
[8,]	0	0	0	0	0	0	0	1	0	0
[9,]	0	0	0	0	0	0	0	0	1	0
[10,]	0	0	0	0	0	0	0	0	0	1

This is in strong contrast to the correlations among the original variables – see the object **P** that we created in step 2 above. This is why we can analyze or use each PC independently of the others.

The PC scores can also be graphed (see ‘The Biplot: Visualizing a PCA’ section below).

How Many Principal Components?

By definition, there are as many principal components as there were variables in the dataset. Each principal component is an axis in the ordination space, with an associated eigenvalue and eigenvector.

Depending on our objectives, we might want to use them all or to focus on a subset of them.

Use Them All ... (Data Rotation)

If we retain all of the principal components, we have rotated the data by expressing the location of each sampling unit in terms of the new synthetic variables created for each principal component. However, we have not altered the dimensionality of the data or ignored any of the data. The rotated data are in the same relative positions as in the original data; all that has changed is the axes along which the data cloud is arrayed.

By rotating a data cloud like this, we ensure that we are looking at that data cloud from the perspective that shows as much variation as possible. This is commonly done, for example, following a NMDS ordination.

... Or Focus on a Few (Data Reduction) ...

When our goal is data reduction, we want to focus on a small number of principal components. However, we also want to account for as much of the variation within the dataset as possible. Since the eigenvalues are arranged in descending order in terms of the amount of variance they account for, the first PCs will often explain most of the variation. Later PCs account for little variation and therefore can be dropped with minimal consequences for interpretation.

There are several ways to determine how many principal components (axes) to consider (Peres-Neto et al. 2005). These ways may give different results.

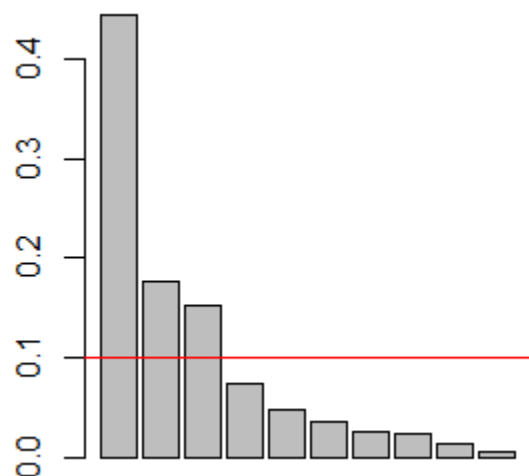
Visually, we can examine a **scree plot**. This is simply the proportion of variance explained by each principal component, with the PCs plotted in order. We can examine this in several ways:

- Look for a sharp bend or change in slope in the proportion of variance explained
- Focus on those principal components that explain more variation than average
- Focus on all principal components that explain more variation than expected, where the expectations are random proportions. This is illustrated below.

We can use `barplot()` to create a scree plot and then superimpose the mean eigenvalue onto that graphic:

```
barplot(S.eigen.prop)

abline(h = mean(S.eigen.prop),
       col = "red")
```



Scree plot showing the proportion of variance explained by each principal component (PC1 on left, PC10 on right) for the ten variables measured on Darlingtonia plants. The red line shows the average value of an eigenvalue for this dataset.

Finally, this decision could be made quantitatively: to focus on as many eigenvalues as necessary to account for 75% (Borcard et al. 2018) or 90% (Crawley 2007) of the total variation.

In our case, it is reasonable to focus on the first three components. There appears to be a break in the scree plot below this, each of these eigenvalues explains more variation than expected, and together they explain 77% of the variance.

... Or Focus on One at a Time

Since the PCs are orthogonal to one another, we can consider them individually. This is helpful when they capture different 'facets' of the responses – see the above interpretation of pitcher size (PC1), pitcher shape (PC2), and leaf width (PC3). We can test whether pitcher size (PC1) relates to some

predictor, without size being influenced by pitcher shape (PC2) or leaf width (PC3). This is illustrated below.

One advantage of this approach is that we do not have to decide which of our measured variables best represents a facet. For example, rather than having to choose one variable as an index of pitcher size, we can analyze the combined effects of all of the variables on this facet.

PCA Functions in R

While it is helpful to see the steps involved in a PCA, it would be slightly laborious to work through them each time you did a PCA. R, of course, contains several PCA functions. They differ slightly in computational method and in the format and contents of the output.

The PCA functions that we will review here are `stats::princomp()` and `stats::prcomp()`. Notably, each of these functions has a default argument that generally needs to be overridden.

Other functions to conduct PCA include `vegan::rda()` and `ade4::dudi.pca()`.

stats::princomp()

The `princomp()` function closely parallels what we did above, using the `eigen()` function to calculate the eigenvalues and eigenvectors. Its usage is:

```
princomp(x,  
  cor = FALSE,  
  scores = TRUE,  
  covmat = NULL,  
  subset = rep_len(TRUE, nrow(as.matrix(x))),  
  fix_sign = TRUE,  
  ...  
)
```

The key arguments are:

- `x` – a matrix or data frame containing the data
- `cor` – whether to use the correlation matrix (`TRUE`) or the covariance matrix (`FALSE`). Default is the latter. However, we noted above that the variables need to be normalized if using the covariance matrix. This step is not built into `princomp()`.
- `scores` – whether to return the score for each observation. Default is to do so (`TRUE`).
- `fix_sign` – whether to set the first loading of each PC to be positive (`TRUE`). This ensures that analyses are more likely to be comparable among runs. If `FALSE`, it is possible for the loadings to have opposing signs from one run to another (note that this does not change the correlations among the variables but simply whether large scores are associated with large or small values of each variable (positive and negative signs, respectively)).

This function returns an object of class 'princomp' which includes numerous components that we calculated in our worked example. These components can each be indexed and called:

- `sdev` – the square roots of the eigenvalues of the variance/covariance matrix. Compare the square of these values to `S.eigen$values`.
- `loadings` – the matrix of loadings (i.e., eigenvectors). This matrix is easier to view than the other loading matrices we've seen, but it has a few quirks. For example, the default is to automatically omit elements < 0.1 (a different cutoff can be specified in the `summary()` function below). Compare to `S.eigen$vectors`.
- `center` – the mean that was subtracted from each element during normalization.
- `scale` – the standard deviation that each element in each column was divided by during normalization.
- `n.obs` – the number of observations (sample units)
- `scores` – the scores for each observation on each principal component. Compare to `PC.scores` (or, equivalently, `scaled data %*% loadings`).
- `call` – the function that was run

Applying this to our example dataset:

```
darl.PCA <- princomp(darl.data, cor = TRUE)
```

We can use `summary()` to obtain a summary of the object we created. Behind the scenes, this function recognizes the class of the object (`princomp`) and applies pre-defined actions based on this class. Knowing this information allows us to find help for it, in the general form `function.class()`. For example, `?summary.princomp` provides arguments to display loadings or not (`loadings = FALSE`) and, if they are displayed, to restrict attention to those above a specified value (default is `cutoff = 0.1`). Once we know these arguments, we can of course adjust them:

```
summary(darl.PCA, loadings = TRUE, cutoff = 0)
```

Importance of components:

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Standard deviation	2.1069368	1.3296011	1.2373322	0.85674840	0.68802564
Proportion of Variance	0.4439183	0.1767839	0.1530991	0.07340178	0.04733793
Cumulative Proportion	0.4439183	0.6207022	0.7738013	0.84720307	0.89454100

	Comp.6	Comp.7	Comp.8	Comp.9
Standard deviation	0.60135218	0.50312405	0.49361594	0.37793991
Proportion of Variance	0.03616244	0.02531338	0.02436567	0.01428386
Cumulative Proportion	0.93070344	0.95601682	0.98038249	0.99466635

	Comp.10
Standard deviation	0.230946945
Proportion of Variance	0.005333649
Cumulative Proportion	1.000000000

Loadings:

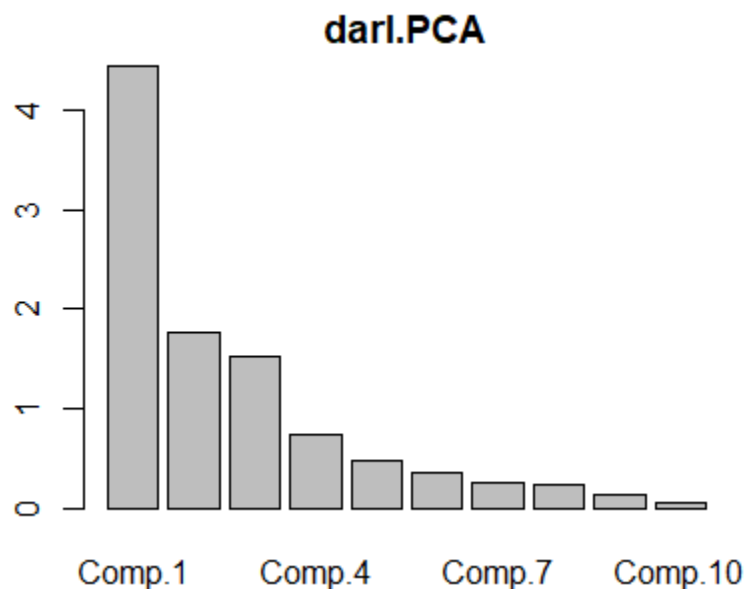
	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6	Comp.7	Comp.8
height	0.297	0.494	0.026	0.122	0.285	0.534	0.062	0.185
mouth.diam	0.390	0.199	-0.178	-0.004	-0.139	-0.418	0.244	0.688
tube.diam	-0.016	0.346	0.635	0.071	0.282	-0.537	0.196	-0.238
keel.diam	-0.197	0.290	0.527	-0.165	-0.626	0.280	-0.157	0.225
wing1.length	0.399	-0.257	0.114	0.072	-0.298	0.034	0.372	-0.121
wing2.length	0.371	-0.284	0.243	0.167	-0.141	0.241	0.357	-0.234
wingsprea	0.272	-0.354	0.344	0.437	0.173	-0.033	-0.614	0.279
hoodmass.g	0.396	0.133	-0.158	-0.232	-0.391	-0.293	-0.439	-0.334
tubemass.g	0.371	0.400	-0.124	0.024	0.132	0.133	-0.189	-0.330

wingmass.g	0.234	-0.249	0.238	-0.821	0.345	0.100	-0.047	0.134
	Comp.9	Comp.10						
height	0.028	0.497						
mouth.diam	0.126	-0.186						
tube.diam	-0.005	0.094						
keel.diam	-0.030	-0.162						
wing1.length	-0.697	0.166						
wing2.length	0.638	-0.165						
wingsprea	-0.070	-0.024						
hoodmass.g	0.201	0.404						
tubemass.g	-0.209	-0.680						
wingmass.g	-0.022	-0.050						

You should be able to match these results to the values obtained in our step-by-step approach above. Focus on one PC at a time and confirm that the magnitude of the loadings is identical but that some PCs have loadings in opposing directions in the two analyses. For example, height has a loading of 0.297 on PC1 in the above table but was -0.297 in our step-by-step analysis. The broad patterns remain the same – variables that increase together in one analysis decrease together in the other – but the interpretation of large or small PC scores will differ. If based on the PCA summarized in the above table, a large value for PC1 would be associated with a larger plant. However, if based on the PCA summarized in the step-by-step analysis, a small value for PC1 would be associated with a larger plant. It is possible for these directions to change from one analysis to another. The current version of `princomp()` includes a `fix_sign` argument that minimizes the occurrence of these differences, but I recommend conducting a PCA and summarizing it in the same session.

To obtain a screeplot of the eigenvalues / principal components we can apply `plot()` or `screeplot()` to an object of class `princomp`:

```
plot(darl.PCA)
```



Scree plot showing the relative importance of each principal component (Comp.1 through Comp.10) for the ten variables measured on *Darlingtonia* plants. The units here are the same as in `S.eigen$values`.

The `vegan` package includes a similarly named `screeplot()` function that can be applied to objects of class 'princomp' or 'prcomp'. It includes a 'broken stick' argument (`bstick`) that compares the variation explained by each PC to a set of 'ordered random proportions' – the idea is that you would retain as many PCs as explain more variation than expected.

stats::prcomp()

The `prcomp()` function uses a different approach, singular value decomposition (the `svd()` function; see Matrix Algebra notes), to obtain the eigenvalues and eigenvectors. Manly & Navarro Alberto (2017) note that this algorithm is considered to be more accurate than that of `princomp()`, though I have not encountered differences between them.

The usage of `prcomp()` is:

```
prcomp(x,  
  retx = TRUE,  
  center = TRUE,  
  scale. = FALSE,  
  tol = NULL,  
  rank. = NULL,  
  ...  
)
```

The arguments are:

- `x` – the matrix or data frame containing the data
- `retx` – whether to return the rotated variables. Default is to do so (`TRUE`).
- `center` – whether to center each variable on zero (i.e., subtract the mean value). Default is to do so (`TRUE`).
- `scale.` – whether to scale each variable to unit variance. Default is to not do so (`FALSE`), though scaling is generally recommended as noted in our worked example (and stated in the help file). Note the '.' in the argument name!
- `tol` – tolerance; components will be deleted if their standard deviation is less than this value times the standard deviation of the first component. Default is to include all variables.
- `rank.` – number indicating how many principal components to report. Default is to report them all. All principal components are still calculated; this just limits attention to those up to the rank specified.

This function returns many of the same components as from `princomp()`, but calls them by different names. Again, these components correspond to those calculated in our worked example and can each be indexed and called:

- `sdev` – the standard deviations of the principal components, also known as the square roots of the eigenvalues of the variance/covariance matrix. Compare squared values to `S.eigen$values`.
- `rotation` – the matrix of loadings (i.e., eigenvectors). Compare to `S.eigen$vectors`.
- `x` – the scores for each observation on each principal component. Compare to `PC.scores` (or, equivalently, `scaled data %*% loadings`).
- `center` – the mean that was subtracted from each element during normalization. Compare to `apply(dar1.data, 2, mean)`.
- `scale` – the standard deviation that each element in each column was divided by during normalization. Compare to `apply(dar1.data, 2, sd)`.

Applying this to our example dataset:

```
darl.PCA2 <- prcomp(darl.data, scale. = TRUE) # Note scale. argument
```

The `summary()`, `print()`, `plot()`, and `screeplot()` functions behave very similarly when applied to an object of class 'prcomp' as they do when applied to an object of class 'princomp' as shown above. For example, the `print()` function returns the variance explained by each principal component along with their loadings:

```
print(darl.PCA2)
```

```
Standard deviations (1, ..., p=10):
 [1] 2.1069368 1.3296011 1.2373322 0.8567484 0.6880256 0.6013522 0.5031240
 [8] 0.4936159 0.3779399 0.2309469

Rotation (n x k) = (10 x 10):
```

	PC1	PC2	PC3	PC4	PC5
height	-0.29725091	-0.4941374	0.02594111	-0.122435477	0.2852089
mouth.diam	-0.38993190	-0.1987772	-0.17804342	0.003956204	-0.1387466
tube.diam	0.01633263	-0.3464781	0.63481575	-0.071324686	0.2822023
keel.diam	0.19728430	-0.2902694	0.52651716	0.164670782	-0.6259843
wing1.length	-0.39911683	0.2574223	0.11432171	-0.071529207	-0.2984514
wing2.length	-0.37099329	0.2836524	0.24337230	-0.166545660	-0.1412989
wingsprea	-0.27156026	0.3541451	0.34374791	-0.437068261	0.1732058
hoodmass.g	-0.39632173	-0.1326148	-0.15831785	0.231712503	-0.3906018
tubemass.g	-0.37122939	-0.4001026	-0.12442151	-0.024231381	0.1319643
wingmass.g	-0.23420022	0.2493993	0.23750541	0.821358528	0.3448007

	PC6	PC7	PC8	PC9	PC10
height	-0.53358014	-0.06213397	0.1851593	-0.028165685	0.49680679
mouth.diam	0.41785432	-0.24376093	0.6877145	-0.125773703	-0.18623421
tube.diam	0.53670782	-0.19570472	-0.2384156	0.005035915	0.09352417
keel.diam	-0.28028754	0.15678491	0.2246203	0.030018781	-0.16165211
wing1.length	-0.03412138	-0.37171835	-0.1206353	0.696871529	0.16630928
wing2.length	-0.24090857	-0.35700239	-0.2344052	-0.638290495	-0.16469409
wingsprea	0.03256432	0.61420874	0.2791219	0.069750311	-0.02361776
hoodmass.g	0.29328788	0.43946368	-0.3340370	-0.200905130	0.40364300
tubemass.g	-0.13345583	0.18870304	-0.3296105	0.209051028	-0.68033777
wingmass.g	-0.09991858	0.04715240	0.1335554	0.022049695	-0.04993778

Using PC Scores

The eigenvalues and loadings (i.e., eigenvectors) are a summary of the PCA. It is important to report these so that the reader knows, for example, which variables were included in the PCA and how important they were for each PC.

Also remember that the meaning assigned to our results may depend on how we interpret those loadings. For example, if someone felt that PC1 more accurately reflected some other aspect of *Darlingtonia* plants, they might disagree with our characterization of it as a measure of pitcher shape.

The key elements that we want to use are the PC scores of the sample units. We can use the PC scores in several ways.

Analyze Each PC Independently

Although PCs were calculated from a set of correlated variables, the PCs themselves are uncorrelated with one another. It therefore can be appropriate to **analyze each PC as a separate response variable**. The PCs may better meet the assumptions of separate univariate hypothesis tests even when it would be inappropriate to do so with the original variables.

For example, we can use PERMANOVA to test for a difference in pitcher size (PC1) among sites. To do so, we'll first combine the scores with the site identity of each plant:

```
darl.PCA.scores <- data.frame(darl.PCA$scores,  
  site = darl$site)
```

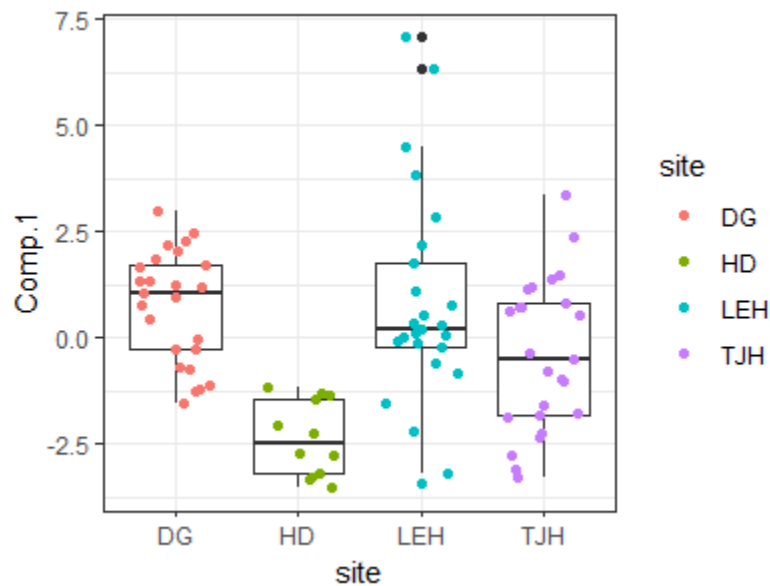
```
adonis2(darl.PCA.scores$Comp.1 ~ site,  
  data = darl.PCA.scores,  
  method = "euc")
```

```
Permutation test for adonis under reduced model  
Terms added sequentially (first to last)  
Permutation: free  
Number of permutations: 999  
  
adonis2(formula = darl.PCA.scores$Comp.1 ~ site, data = darl.PCA.scores, method = "euc")  
      Df SumOfSqs      R2      F Pr(>F)  
site    3   100.53 0.26031 9.7362 0.001 ***  
Residual 83   285.68 0.73969  
Total    86   386.21 1.00000  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This test indicates that pitcher size differs among the sites. We could follow this test with pairwise contrasts to determine which sites differ in 'pitcher size'.

We could also see these differences visually:

```
ggplot(data = darl.PCA.scores, aes(x = site, y = Comp.1)) +  
  geom_boxplot() +  
  geom_jitter(aes(colour = site), width = 0.3, height = 0) +  
  theme_bw()
```



Scores for PC1 (pitcher size), by site. Larger values correspond to larger plants, and smaller values correspond to smaller plants (see loadings table to confirm this, and see the biplot below).

Since PCs are orthogonal, we could conduct identical tests for differences among sites with respect to PC2 (pitcher shape) and PC3 (leaf width). Tests of one PC are **always** independent of tests of other PCs produced by that PCA.

Reduce Dimensionality of Explanatory Variables

Sometimes we have multiple highly correlated explanatory variables. Rather than having to choose one, we can apply a PCA to this set of variables and thereby **summarize them in a much smaller number of uncorrelated variables** (ideally, one or two). These uncorrelated PC scores can then be used as explanatory variables in subsequent analyses.

One potential criticism of this approach is that the PCs are more abstract than the set of variables from which they are calculated.

Haugo et al. (2011) provide an example of this approach. The authors examined vegetation dynamics within mountain meadows from 1983 to 2009. One factor influencing the meadows is tree encroachment and associated changes in microenvironment. Tree influence could be quantified in many ways, including tree cover, homogeneity of tree cover, tree density, and basal area. Furthermore, these variables could be considered in terms of both their initial (1983) value and the change in that variable (i.e., 2009 value – 1983 value). A PCA was applied to all of these variables. PC1 explained 37% of the total variation and correlated with initial tree structure (tree cover, homogeneity of tree cover, and basal area). PC2 explained 26% of the total variation and correlated negatively with measures of change in tree structure (change in density, basal area, and cover). PC1 and PC2 were then used as explanatory variables, along with others, to understand changes in the richness and cover of groups of species such as those that are commonly associated with forest and meadow habitats.

The Biplot: Visualizing a PCA

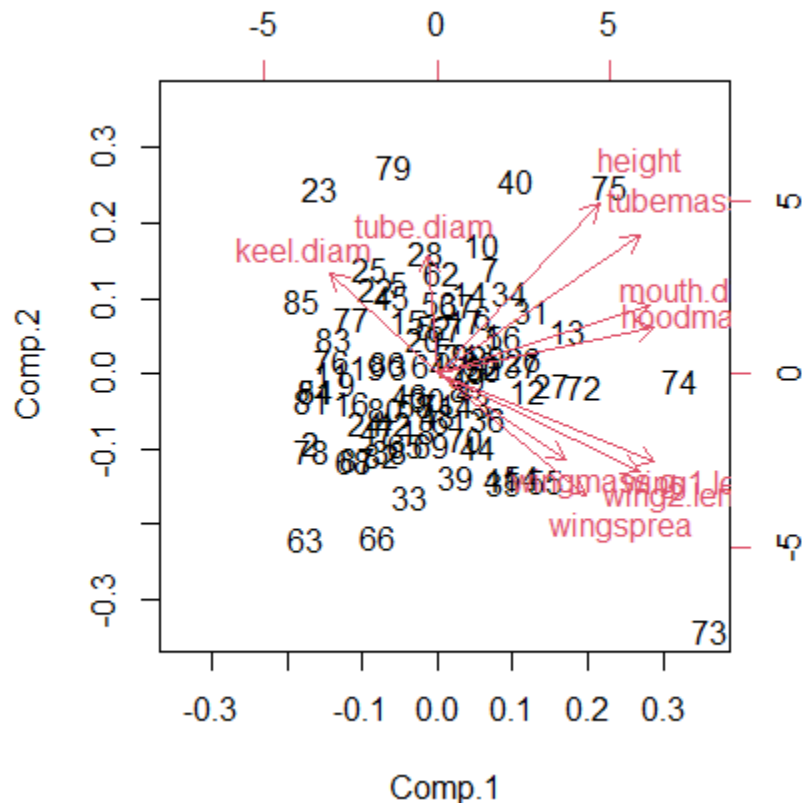
It is common to focus visually on two PCs because we can easily plot them as a scatterplot. When this graphic is overlaid with vectors associated with the variables, it is known as a **biplot**.

To focus on more than two PCs, we can plot combinations of the PCs. For example, if we wanted to graphically show three PCs, we could plot PC1 vs. PC2, PC1 vs. PC3, and PC2 vs. PC3.

Base R Graphing

When applied to a 'princomp' or 'prcomp' object, the `biplot()` function will produce a biplot:

```
biplot(darl.PCA)
```



Biplot of the first two principal components from a PCA of the Darlingtonia plant data. Each plant is represented by its row number. The red vectors point in the directions in which variables increase most strongly.

This kind of graphic is common in ordinations. Some comments about interpreting biplots:

- We focus on the first two PCs because, by definition, they explain more of the variation than any other PCs. To create this in our step-by-step introduction to PCA, we simply needed to graph the first two PCs:
`plot(PC.scores)`

- Each individual data point (row in the sample × species matrix) is represented by its row number. We could customize this plot to display them in other ways, such as by having different symbol shapes and colors for different levels of a grouping variable. See below for an example.
- The scores on each PC axis are centered on zero because of the normalizing that occurred during the PCA.
- The axes on the bottom and left-hand side of the graph show the values of the first two PCs. It is common practice to report the amount of variation explained by each axis in the axis title. For example, this code would change the axis labels of the biplot:
`biplot(dar1.PCA, xlab= "PC1 (44.4%)", ylab= "PC2 (17.7%)")`
- A second coordinate system is shown on the top and right-hand side. According to Everitt & Hothorn (2006, p. 222), this displays the first two loadings associated with these variables – though I haven't verified this.

The vectors (arrows) are an important aspect of a biplot:

- Each vector corresponds to a variable, and points in the direction in which that variable is most strongly linearly correlated.
- Variables that are perfectly correlated with an axis are parallel to it.
- Variables that are uncorrelated with an axis are perpendicular to it.
- Each vector can be thought of as the hypotenuse of a triangle with sides along the two axes of the graph. The length of these sides is proportional to the loading for that variable with those PCs.
- The angle between vectors shows the strength and direction of the correlation between those variables. You should be able to find examples of each of these cases in the above biplot:
 - Vectors that are very close to one another represent variables that are strongly positively correlated
 - Vectors that are perpendicular represent variables that are uncorrelated
 - Vectors that point in opposite directions represent variables that are strongly negatively correlated

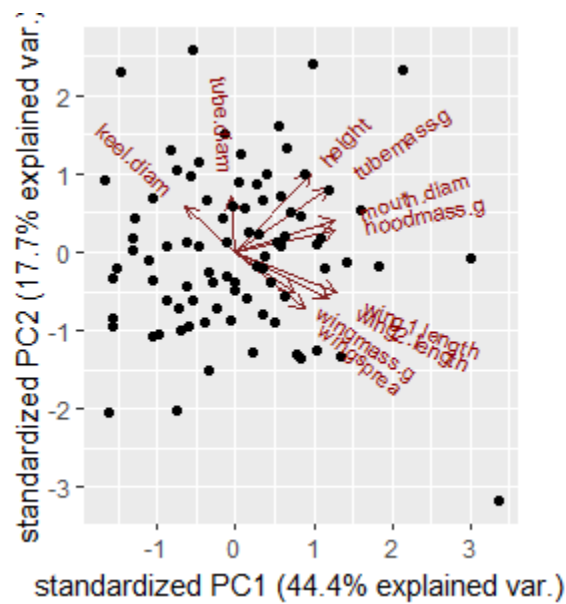
The ggbiplot Package

The `ggplot2` package does not have a built-in capability to draw biplots, but the `ggbiplot` package allows their creation using the `ggplot2` approach. It is available from a GitHub repository:

```
devtools::install_github("vqv/ggbiplot")

library(ggbiplot)

ggbiplot(dar1.PCA)
```

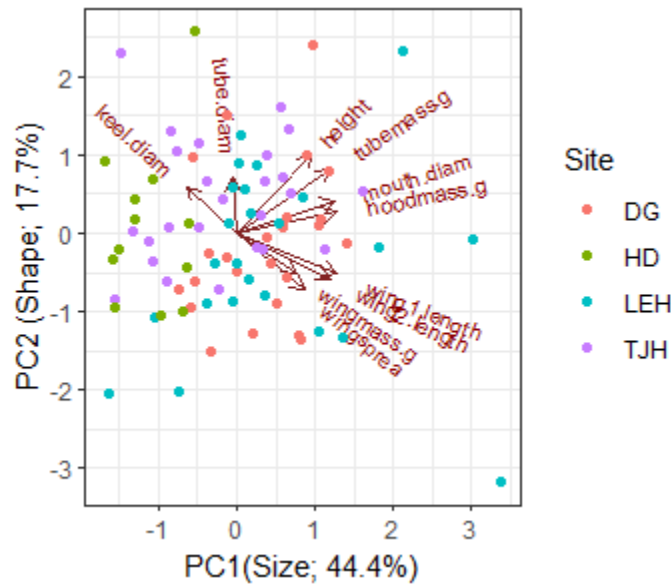


Biplot of the first two principal components from a PCA of the Darlingtonia plant data. Each plant is represented by a black symbol. The red vectors point in the directions in which variables increase most strongly.

Compare this graphic with the biplot produced above with base R plotting capabilities.

The `ggbiplot()` function can be combined with other standard `ggplot2` functions. For example:

```
ggbiplot(darl.PCA) +
  geom_point(aes(colour = darl$site)) +
  labs(colour = "Site", x = "PC1(Size; 44.4%)", y = "PC2 (Shape; 17.7%)") +
  theme_bw()
```



Biplot of the first two principal components from a PCA of the Darlingtonia plant data. Each plant is represented by a symbol, with colors corresponding to the four sites. The red vectors point in the directions in which variables increase most strongly

I've changed the theme and the axis labels, and color-coded each plant by the site it came from. However, remember that site identity was not part of the PCA.

Conclusions

PCA identifies the best linear fit to a set of variables and therefore is intended for data with linear relationships among variables. It assumes that these variables are continuously distributed and normally distributed (recall the importance of normalization in our step-by-step example). The assumption of multivariate normality is particularly important if you are going to use a PCA to make statistical inferences rather than just to describe a data set.

PCA is appropriate under certain scenarios:

- if the sample units span a short gradient (i.e., beta diversity is low; little turnover among sample units)
- if the data matrix is not sparse (i.e., does not contain many nonzero values)
- when variables are highly (linearly) correlated with one another

Examples where PCA is often appropriate include:

- LiDAR data (multiple highly correlated and continuously distributed variables)
- fuels data
- morphological data
- physiological data

PCA has been used for community-level data (i.e., sample unit × species matrices) in the past, but these applications are considered unsuitable (Beals 1971; Minchin 1987; Legendre & Legendre 2012). Problems when applying PCA to this type of data include:

- PCA works best with highly correlated response variables, but the abundances of many species are only weakly correlated with one another.
- Sensitive to outliers. If data are highly skewed, the first few principal components will separate the extreme values from the rest of the sample units.
- “The quality of ordination by PCA is completely dependent on how well relationships among the variables can be represented by straight lines” (McCune & Grace 2002, p. 116).
- Need more sample units than variables measured (technicality; I’m not sure how important this really is).
- Implicitly uses the Euclidean distance measure
- May yield horseshoe effect (artificial curvature) in second and higher axes if sample units span a long gradient. This occurs because shared zeroes are interpreted as an indicator of a positive relationship, and because two species may be positively related to each other at some points and negatively related at other points. See Fig. 19.2 in McCune & Grace (2002) for an example of this.
- ‘Axis-centric’? (term borrowed from Mike Kearsley at Northern Arizona University) – we tend to assign more meaning to axes than may be warranted.
- Tendency to interpret proximity of apices of variables (vectors) rather than the angles separating them.

Although PCA is inappropriate for analyzing community-level data, there are certain applications in which it is appropriate even with these data. In particular, PCA can be used to rotate the coordinates from another ordination so that the ordination solution is expressed in such a way that each axis explains as much of the remaining variation as possible. When doing so, all principal components are retained and thus the dimensionality of the ordination solution is unchanged; the rotated data are in the same relative positions as in the original ordination solution.

Not considered here are alternatives such as Multiple Correspondence Analysis (MCA), which can handle categorical variables (Greenacre & Blasius 2006).

References

- Beals, E.W. 1971. Ordination: mathematical elegance and ecological naivete. *Journal of Ecology* 61:23-35.
- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Crawley, M.J. 2007. *The R book*. John Wiley & Sons, Hoboken, NJ.
- Everitt, B.S., and T. Hothorn. 2006. *A handbook of statistical analyses using R*. Chapman & Hall/CRC, Boca Raton, LA.
- Goodall, D.W. 1954. Objective methods for the classification of vegetation. III. An essay in the use of factor analysis. *Australian Journal of Botany* 2:304-324.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.
- Greenacre, M., and J. Blasius. 2006. *Multiple correspondence analysis and related methods*. Chapman & Hall/CRC, London, UK.

Haugo, R.D., C.B. Halpern, and J.D. Bakker. 2011. Landscape context and tree influences shape the long-term dynamics of forest-meadow ecotones in the central Cascade Range, Oregon. *Ecosphere* 2:art91.

Hotelling, H. 1933. Analysis of a complex of statistical variables in principal components. *Journal of Experimental Psychology* 24:417-441, 493-520.

Legendre, P., and L. Legendre. 2012. *Numerical ecology*. 3rd English edition. Elsevier, Amsterdam, The Netherlands.

Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Minchin, P.R. 1987. An evaluation of the relative robustness of techniques for ecological ordination. *Vegetatio* 69:89-107.

Pearson, K. 1901. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine, Sixth Series* 2:559-572.

Peres-Neto, P.R., D.A. Jackson, and K.M. Somers. 2005. How many principal components? stopping rules for determining the number of non-trivial axes revisited. *Computational Statistics & Data Analysis* 49:974-997.

Summerville, K.S., C.J. Conoan, and R.M. Steichen. 2006. Species traits as predictors of lepidopteran composition in restored and remnant tallgrass prairies. *Ecological Applications* 16:891-900.

Media Attributions

- [Darlingtonia.californica_Jepson](#)
- [PCA.barplot2](#)
- [darl.screepplot](#)
- [PCA.pitcher.size.gg](#)
- [PCA.biplot](#)
- [PCA.biplot.gg](#)
- [PCA.biplot.gg2](#)

37. NMDS

Learning Objectives

- To demonstrate how the NMDS algorithm works.
- To consider when NMDS is an appropriate ordination technique.
- To demonstrate how the fit of a NMDS ordination can be assessed.
- To continue creating graphical images of ordinations.

Reading (Recommended)

Clarke (1993, p. 117-126)

Key Packages

```
require(vegan, tidyverse)
```

Introduction

Non-metric MultiDimensional Scaling (NMDS) is a **distance-based ordination technique**. Because it focuses on the distance matrix, it is very flexible – any distance measure can be used. As usual, the data matrix (n sample units \times p species) is converted into an $n \times n$ distance matrix (or, more generally, a dissimilarity matrix).

Note: you will likely encounter a variety of abbreviations for Non-metric MultiDimensional Scaling (NMDS). People variously refer to it as non-metric MDS or NMS or MDS. I do not recommend this last abbreviation for NMDS, as it is also used to for metric MultiDimensional Scaling (aka PCoA), which we consider separately.

Characteristics of NMDS

NMDS was first popularized in the psychological literature (Shepard 1962a,b; Kruskal 1964a,b). It can be used to obtain an ordination from any distance measure. As we've discussed numerous times, the distance matrix summarizes the differences among each pair of sample units in terms of any number of response variables.

NMDS generally does a good job at representing complex multi-dimensional data in a small number of dimensions (usually 2 or 3). As a result, it is often used for data reduction and visualization.

NMDS is sometimes used simply for data reduction. For example, Haugo et al. (2011) examined vegetation dynamics within mountain meadows from 1983 to 2009. They conducted a NMDS ordination of the meadow communities in 1983, and determined that the coordinates in this ordination space related to landscape context (landform, hydrology, and elevational zone). They therefore used these coordinates as explanatory variables in a subsequent analysis of how the vegetation changed between 1983 and 2009.

More commonly, NMDS is used for data visualization (which often requires data reduction). Most papers that use NMDS will show one or more ordinations produced by this technique.

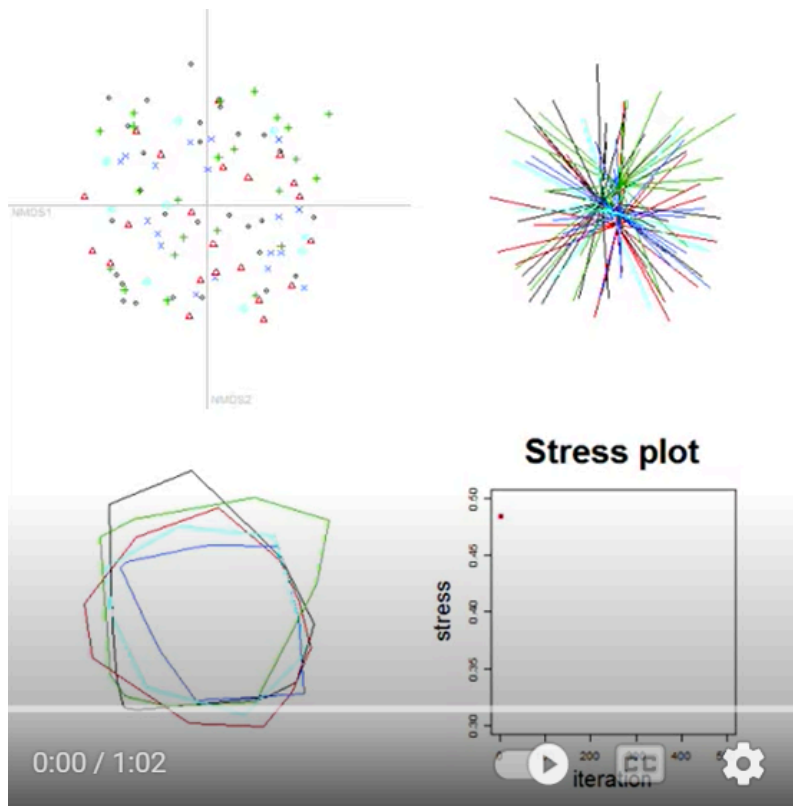
Techniques such as PCA can sometimes be used in other ways, such as to rotate a data cloud so that axes are ordered in declining order of importance. I cannot think of any situation where we would be interested in using NMDS for this type of application.

NMDS differs in several important ways from other ordination techniques:

- As noted above, NMDS is a distance-based technique.
- NMDS does not plot the distances *per se* among sample units. Instead, it attempts to **plot sample units in such a way that the distances among sample units in the ordination space are in the same rank order as the distances among sample units as measured by the original distance matrix**. In other words, it attempts to choose coordinates so that sample units that are similar in 'real life' are close together in the ordination space and sample units that are dissimilar in 'real life' are far away from one another in the ordination space.
- It can proceed even if some distances are missing from the dissimilarity matrix.
- It does not seek to maximize the variability associated with individual axes of the ordination. As a result, **the axes of an NMDS ordination are entirely arbitrary**, and plots may be freely rotated, centered, or inverted to increase interpretability.
- It is **iterative** – the positions of the sample units in ordination space are adjusted through a series of steps.
- Due to its iterative nature, it is **computationally intensive** to run with very large samples.
- The **user must specify how many dimensions** are desired for the solution. Solutions can differ depending on the number of dimensions selected: a 2-D solution may not be the same as the first two dimensions of the 3-D solution.
- There is no single solution.

NMDS In Action

It's more difficult to discuss the theory behind NMDS than to see it in action. Let's explore the process of doing a NMDS as illustrated in this video. Here's a screenshot from the start of the video:



Screen shot from the start of a video demonstrating the iterative nature of NMDS. The upper left, upper right, and lower left images show three ways of visualizing the same set of data. The sample units belong to five groups (colors), but group identity is not part of the NMDS algorithm. The lower right image shows the stress (a measure of how well the ordination aligns with the real data) at the start of this process.

This image contains four graphics:

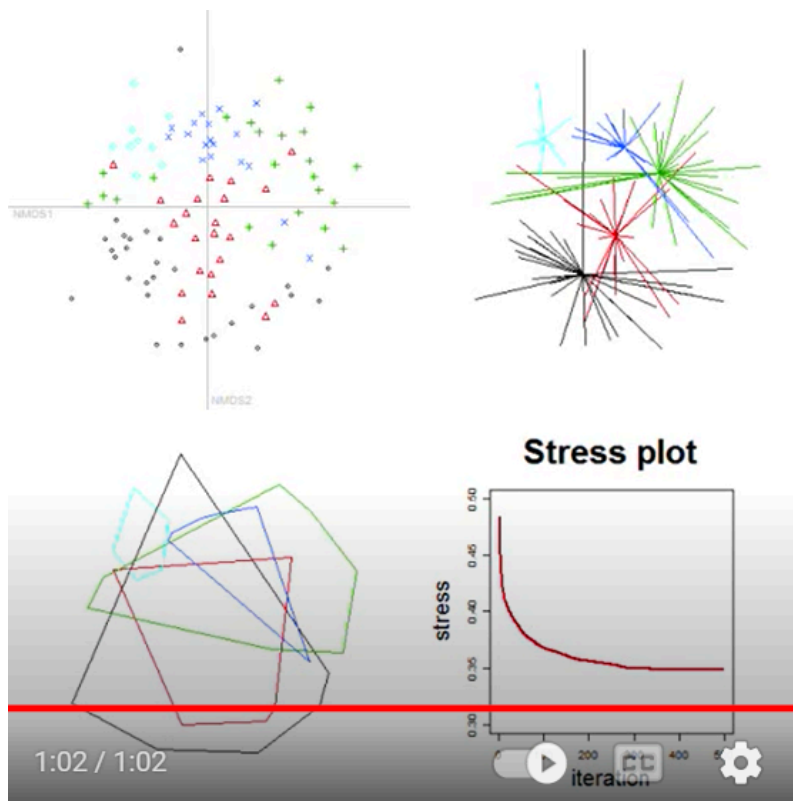
- Top left – the ordination space, with each sample unit shown as a point. Symbol shape and color distinguish the groups for illustration purposes.
- Top right – the ordination space, with each sample unit connected to the centroid of the group to which it belongs. This type of image is called a 'spider'.
- Bottom left – the ordination space, with a perimeter (hull) encompassing all of the sample units from each group.
- Bottom right – a 'stress plot' showing how model fit, as measured by stress (see below), changes iteratively.

It is important to note that group identity is shown for illustration purposes. The NMDS algorithm proceeds on the basis of the distance matrix, and is not influenced by group identity.

Initially, the sample units are positioned at random starting coordinates in the 2-D ordination space. As a result, the points are interspersed and the spiders and hulls of the different groups are indistinguishable.

The NMDS algorithm involves calculating the distances between sample units in the ordination space, comparing these distances to the distances based on the actual responses, and iteratively adjusting the coordinates of the sample units in the ordination space to increase the agreement between these two sets of distances (i.e., to lower the stress).

Here is a screenshot from the end of this process – in this case, after 500 iterations:



Screen shot from the end of a video demonstrating the iterative nature of NMDS. The upper left, upper right, and lower left images show three ways of visualizing the same set of data. The sample units belong to five groups (colors), but group identity is not part of the NMDS algorithm. The lower right image shows how the stress was reduced (i.e., the ordination increasingly aligned more strongly with the real data) during this process.

Observe how sample units from different groups tend to occur in different locations within the two-dimensional ordination space. This separation was based on the values in the distance matrix, not on the group identities. Group identity was used to distinguish observations within this example, but is not 'known' to the NMDS algorithm.

NMDS in R (`vegan::metaMDS()`)

NMDS can be conducted using several functions in R. We will use the functions available in the `vegan` package; others include `MASS::sammon()`, `ecodist::nmds()`, and `smacof::mds()`. Functions often differ in argument names and may differ in other aspects (e.g., choice of initial coordinates), so be sure to check the appropriate help files if using other functions.

The `vegan` package uses `monoMDS()` as its 'workhorse' NMDS function. The arguments of this function allow us to control the details of the NMDS ordination. Additional functions process the data file before it can be used by `monoMDS()` and process the results afterwards. These individual functions can also be called directly if you want to retain maximal control of all aspects of the NMDS process. However, it is more common to use a 'wrapper' function, `metaMDS()`, which has

the advantage of also calling these other functions before and after `monoMDS()`. This function is structured to follow the recommendations of Minchin (1987). For example, `metaMDS()`:

- Allows a data matrix to be specified (in comparison, `monoMDS()` requires a distance matrix)
- Can automatically decide whether/how to transform the data
- Can automatically decide which type of distance measure to use

When calling other functions (e.g., `vegdist()` to convert a data matrix to a distance matrix); the defaults of those functions apply unless overwritten using arguments within `metaMDS()`.

The help file for `metaMDS()` shows the many arguments that are part of its usage. Here, I have organized the key arguments within the steps of a NMDS ordination, including the wrap-around functions. If arguments have defaults, they are identified here as well. I have highlighted key settings in **bold**.

Pre-NMDS (Initial) Steps

Begin with **comm**, a community data matrix (n sample units \times p species) or a $n \times n$ distance matrix. If a distance matrix is specified, the rest of the pre-NMDS steps are skipped.

Transform data matrix if `autotransform = TRUE`. The `sqrt()` transformation is applied if the maximum value in **comm** is ≤ 50 and `wisconsin()` standardization is applied if the maximum value in **comm** is > 9 . However, as noted in the help file, these criteria are arbitrary. To ensure clarity and control of the analysis procedure, I recommend doing desired transformations and/or standardizations before using `metaMDS()`. I therefore recommend setting **`autotransform = FALSE`**.

Convert the data matrix to a distance matrix, using an appropriate distance measure. This calls `vegdist()` and uses its default of the Bray-Curtis distance (**`distance = "bray"`**), but any distance measure available in that function can be called here. If you want to use a distance measure that is not available within `vegdist()`, calculate the distance matrix using another function and specify it as **comm**.

Choose which **engine** (function) to conduct the NMDS with. The default, **`engine = "monoMDS"`**, is assumed in these notes. The alternative, **`engine = "isoMDS"`**, is included here for backward compatibility with early versions of `vegan` that did not contain `monoMDS()`.

Choose how to deal with long environmental gradients. Recall that the Bray-Curtis distance measure (and related measures) have an upper bound of 1 if two sample units do not share species. If the compositional data span a long environmental gradient such that many sample units do not share species, it can be difficult to identify patterns as some sample units are completely different from many other sample units. There are two ways to deal with this, depending on the engine used:

- If the default engine is being used (**`engine = "monoMDS"`**), the **`weakties`** argument allows observations with identical dissimilarities to have different fitted distances (**`weakties = TRUE`**). In other words, setting **`weakties = TRUE`** allows observations to be positioned at different coordinates within the ordination space even though they are the same distance apart in the distance matrix. Setting **`weakties = FALSE`** forces observations that are the same distance apart in the distance matrix to also be the same distance apart in ordination space.
- If the alternative engine is being used (**`engine = "isoMDS"`**), extended dissimilarities can be calculated using the **`noshare`** argument, which calls the `stepacross()` function. Extended dissimilarities incorporate paths that include 'intermediate' sample units – those that share some species with two sample units that do not share species directly (De'ath 1999). Using extended dissimilarities has implications that have not been fully explored, including the fact that the distance measure no longer has a constant upper bound. The argument default is **`noshare = (engine == "isoMDS")`** – this looks confusing but is simply **`TRUE`** if `isoMDS` has

been specified and `FALSE` otherwise.

NMDS Steps

1. Specify how many dimensions (**k**) you want the solution to contain. Note that you will get a different result if you use different values of **k**! In the help files, it is recommended that you have at least twice as many sample units as dimensions (though this recommendation is trivial in most situations).

2. Assign a starting configuration (**y**) to sample units. This can be done many ways:

- Randomly assign starting coordinates to sample units. This is the default in `monoMDS()`.
- Use results of another ordination (e.g., PCA, PCoA) as starting coordinates. For example, set `y = cmdscale(d, k)` to do a PCoA ordination of `d` in `k` dimensions. This is the default in `metaMDS()`.
- Use the `previous.best` argument to begin with the best solution from a previous run of `metaMDS()`.
- If data are spatially structured, use the geographic position of each sample unit as its starting coordinates.
- Compute NMDS using (`k + 1`) dimensions. Use final coordinates from this ordination as the starting coordinates in a run using `k` dimensions.

3. Calculate Euclidean distances among sample units in the ordination space.

4. Compare the fitted distances from the ordination space with the distances in the original dissimilarity matrix (see 'Evaluating Fit' section below). The calculated goodness-of-fit of a regression between these distances is called the stress. Stress is usually calculated using monotone regression (i.e., based on the ranks of the data); this is done by setting `model = "global"`. The other options ("`local`", "`linear`", "`hybrid`") allow you to use this function to conduct other types of NMDS. Stress is described in the 'Stress' section below. `monoMDS()` reports stress as a proportion (i.e., bounded between 0 and 1) and provides two ways to calculate it (1 or 2); `stress = 1` is the standard option and the default.

5. Determine whether the fit is 'good enough'. `monoMDS()` uses three different stopping criteria to determine whether changes in configuration are improving fit. In my experience, these stopping criteria generally do not require adjustment. They are:

- `smin = 1e-4`: stop when stress drops below this value
- `sfgrmin = 1e-7`: stop when 'minimum scale factor' (I'm not sure how this is calculated) is below this value
- `sratmax = 0.999999`: stop when stress ratio exceeds this value

6. If none of the stopping criteria have been met, improve the configuration by moving sample units in a direction that will reduce the stress. This is done using a method called 'method of steepest descent'. Basically, the idea is to shift coordinates in the manner that will cause the greatest reduction in the stress. The amount to move the configuration is determined by the step length; this value is recalculated after each iteration and gets smaller as the stress declines. McCune & Grace (2002, p. 128) describe a landscape analogy for NMDS that may be helpful for understanding this.

7. Repeat from step 3 until either:

- Convergence has been achieved.
- The maximum number of permitted iterations (`maxit = 200`) have been performed. McCune & Grace (2002, Table 16.3) recommend a maximum of 400 iterations for configuration adjustments.

8. A single NMDS run identifies a local minimum, but this is not necessarily the global minimum. New runs help assess this. Each new run involves creating new random initial coordinates (step 2) and rerunning steps 3-7. The stress associated with the final configuration of the new run is compared to that from the initial run. If the new run resulted in a smaller stress, its configuration is saved as the new final configuration. This is repeated at least **try = 20** times, and up to **trymax = 20** times, or until the convergence criteria are met. McCune & Grace (2002, Table 16.3) recommend repeating this 40 times, though we can of course do more.

The `trace = 1` argument causes the final stress value obtained from each run to be reported. Note that this can also be specified as `trace = TRUE`.

The default is to not display the results graphically (`plot = FALSE`). Changing this to `plot = TRUE` produces Procrustes overlay plots comparing each new solution with the previous best solution. We'll discuss Procrustes analysis [later](#).

9. Adjust final configuration:

- Center solution so that the centroid is at the origin.
- Rotate axes using PCA (**pc = TRUE**). All *k* axes are kept, so this process rotates the data but does not alter its dimensionality. By definition, the resulting data cloud is oriented so that the first dimension expresses as much variation as possible, the second as much of the remaining variation as possible, etc. Alternatively, the final configuration can be rotated to align with an environmental variable using `MDSrotate()` – we'll illustrate this [later](#).
- Re-scale ordination axes to 'unit root mean squares' (`scaling = TRUE`). Note that we generally do not pay much attention to the scales of the axes, however.

Adopt the coordinates of each sample unit from the final configuration as its coordinates in the *k*-dimensional ordination space.

Post-NMDS Steps

Calculate species scores as the weighted average of the coordinates of the plots on which those species occurred. This argument (`wascores = TRUE`) simply calls `wascores()`. See the 'Species Scores' section below for details.

Oak Example

Let's conduct a NMDS on the oak plant community dataset.

Use the `load.oak.data.R` script to load and make initial adjustments to the oak plant community dataset. Although this script calculates a distance matrix, we will work with the intermediate object `oak1`, which contains abundances of the 103 most abundant species, each relativized by its maxima. Using this data matrix allows us to see how to control the initial (pre-NMDS) options within `metaMDS()`.

```
source("scripts/load.oak.data.R")
```

Conduct a NMDS. For completeness, I will specify the key arguments here even though I want to use their default settings. Doing so avoids potential issues if the default would change, for example.

To ensure results are repeatable, we'll set the random number generator immediately before doing

so. You're welcome to explore the extent to which altering the random number generator changes the results.

```
set.seed(42)

z <- metaMDS(comm = Oak1,
  autotransform = FALSE,
  distance = "bray",
  engine = "monoMDS",
  k = 3,
  weakties = TRUE,
  model = "global",
  maxit = 300,
  try = 40,
  trymax = 100)
```

The resulting object is of class 'metaMDS'. The `summary()` for an object of this class is not that informative, but we can use `print()` to see a summary of what was done:

```
print(z)
```

```
Call:
metaMDS(comm = Oak1, distance = "bray", k = 3, try = 40, trymax = 100, engine = "monoMDS")

global Multidimensional Scaling using monoMDS

Data:      Oak1
Distance: bray

Dimensions: 3
Stress:     0.1641748
Stress type 1, weak ties
Best solution was repeated 7 times in 40 tries
The best solution was from try 0 (metric scaling or null solution)
Scaling: centring, PC rotation, halfchange scaling
Species: expanded scores based on 'Oak1'
```

We can also use `str()` to explore the object and identify particular items of interest. Those items include:

- Stress value of solution – `z$stress`
- Coordinates of each sample unit in the k -dimensional ordination space – `z$points`. These coordinates can also be extracted using `scores(z)`.
- Coordinates of each species in the k -dimensional ordination space – `z$species`. Only available if `wascores = TRUE` during analysis.
- The distances between sample units in the original distance matrix – `z$diss`.
- The distances between sample units in the ordination space – `z$dist`.

Stress

Stress can be calculated using several formulae (McCune & Grace 2002, p. 126-127). However, stress formulas generally yield similar configurations. In R, stress is reported as a proportion (0-1) or a percent (0-100). The default formula in `monoMDS()` is:

$$S = \sqrt{\frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n (d_{ij} - \tilde{d}_{ij})^2}{\sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij}^2}}$$

where d_{ij} is the distance in the original dissimilarity matrix and \tilde{d}_{ij} is the fitted distance in ordination space. This formula is what is used when `stress = 1`; when stress is calculated using the alternative formulation (`stress = 2`), the denominator is adjusted; see `?monoMDS` for details.

As noted in the last section, the stress achieved by the final solution of a NMDS is part of the resulting object:

`z$stress`

0.1641748

Is this good? Clarke (1993, p. 126) suggests the following rules of thumb for stress values:

Stress (0-1 scale)	Interpretation
< 0.05	Excellent representation with no prospect of misinterpretation
< 0.10	Good ordination with no real risk of drawing false inferences
< 0.20	Can be useful but has potential to mislead. In particular, shouldn't place too much reliance on the details
> 0.20	Could be dangerous to interpret
> 0.35	Samples placed essentially at random; little relation to original ranked distances

While these rules of thumb are useful, they're guidelines rather than absolutes. In particular, the stress of a given solution is a function of:

- Sample size – “the greater the number of samples, the harder it will be to reflect the complexity of their inter-relationship in a two-dimensional plot” (Clarke 1993, p. 125).
- Nature of data – is stress attributable to all points or to a single one that isn't easily placed?

One way to explore the effect of individual points is with the `goodness()` function:

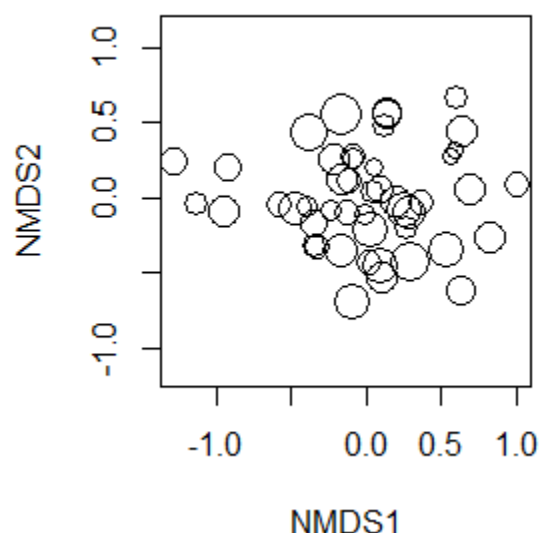
```
gof <- goodness(object = z)
```

The resulting vector summarizes the proportion of the overall variance that is explained by each sample unit. An extremely large value would be an indication that a sample unit is causing a poor fit in the ordination – perhaps there is something unique about it? I've also had times when a data entry error caused one sample unit to appear hugely different from all of the others. We can visualize these statistics:

```
plot(z, display = "sites", type = "none")
```

```
points(z, display = "sites", cex = 2*gof/mean(gof))
```

Note that the `cex` (character expansion) argument looks complicated but is just sizing each symbol to be proportional to its value relative to the mean of them all. Sample units shown with larger circles account for more of the variation in the overall fit of these data.



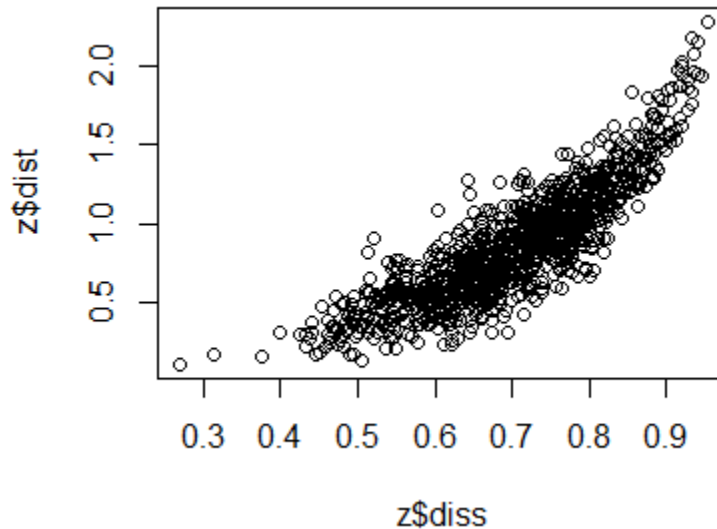
First two dimensions of a three-dimensional NMDS solution for the oak plant community dataset (47 stands). Symbol size is proportional to the amount of variance associated with each sample unit. Stress = 0.16.

Evaluating Fit

The fit of a NMDS ordination can be assessed by plotting the original dissimilarities (`z$diss`) against the (Euclidean) ordination distances (`z$dist`). The number of distances and dissimilarities increases rapidly with the number of sample units (N): recall that each matrix contains $N(N - 1)/2$ pairwise elements. Also, the process of comparing two distance matrices should remind you of a Mantel test.

A plot of dissimilarities vs. ordination distances is known as a **Shepard plot** (see Fig. 2 in Clarke 1993). It can be created manually:

```
plot(z$diss, z$dist)
```

Shepard plot of the oak plant community dataset (47 stands), showing Bray-Curtis dissimilarities between pairs of stands (z\$diss) and the Euclidean distances between the same pairs of stands in the 3-dimensional ordination space (z\$dist).

You can calculate the distances directly from the data to verify that the mapped values in the Shepard plot are from the original data and the coordinates in the ordination space:

```
#plot(vegdist(Oak1), dist(scores(z, "sites"))) #compare to above
```

Shepard plots can also be created using the `stressplot()` function in `vegan`. One advantage of this over the manual approach is that it also adds a fit line (a non-linear step function) and two 'correlation-like' statistics:

- Non-metric or monotonic fit – calculated as $1 - z\$stress^2$.
- Linear fit – squared correlation between fitted values and ordination distances.

The usage of `stressplot()` is:

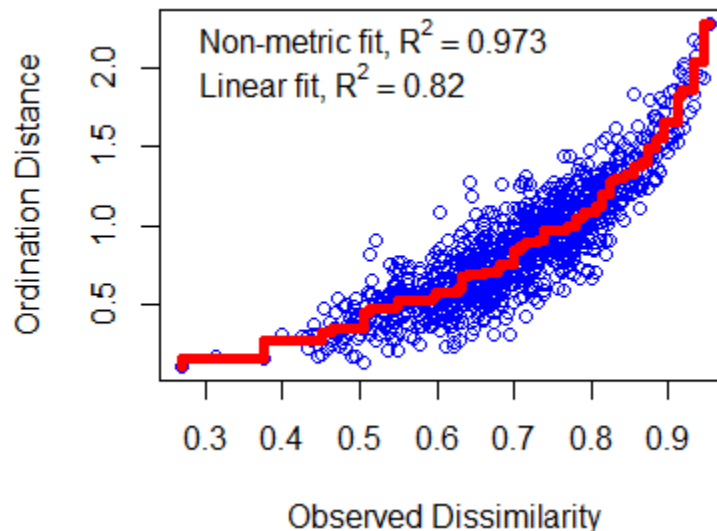
```
stressplot(object,
  dis,
  pch,
  p.col = "blue",
  l.col = "red",
  lwd = 2,
  ...
)
```

The arguments are:

- `object` – the output of a `metaMDS()`, `monoMDS()`, or `isoMDS()` function.
- `dis` – dissimilarities. Only required if `object` is from `isoMDS()`.
- `pch` – plotting character for points. Default is dependent on the number of points.
- `p.col` – point color. Default is blue.
- `l.col` – line color. Default is red.

- lwd – line width.

```
stressplot(object = z, lwd = 5) # Thicker line
```



Shepard plot of the oak plant community dataset (47 stands), showing Bray-Curtis dissimilarities between pairs of stands (Observed Dissimilarity) and the Euclidean distances between the same pairs of stands in the 3-dimensional ordination space (Ordination Distance).

Graphing the Final Configuration

A unique element of graphing a NMDS ordination is that the axes cannot be interpreted like those in a PCA – instead, the focus is on the data points themselves, and particularly on their orderings relative to one another. Therefore, **NMDS ordinations are often plotted without units or labels on the axes so that people do not try to interpret them.**

Obtaining the Ordination Coordinates

To graph a NMDS ordination (or any other object!), we need to know where the coordinates are stored. As noted above, the `str()` function is a good starting point. In this case, the coordinates are saved as `points` in the object.

```
z$points %>% head()
```

	MDS1	MDS2	MDS3
Stand01	-0.39411569	-0.05254510	0.3604700
Stand02	-0.34549234	-0.17365891	0.2593316

```
Stand03 -0.23802128 -0.08292349  0.1002232
Stand04 -0.09426964  0.27745268  0.1299250
Stand05  0.82431684 -0.27299339 -0.4049859
Stand06 -0.47766863 -0.07318759 -0.2589466
```

These coordinates are the positions of each sample unit in the ordination space. Points near one another are similar in their multivariate response, while points far from one another are different in their multivariate response.

We can either call the coordinates directly, save them to a new object, or combine them with the raw data or the other explanatory variables. Here, I'll save them to their own object:

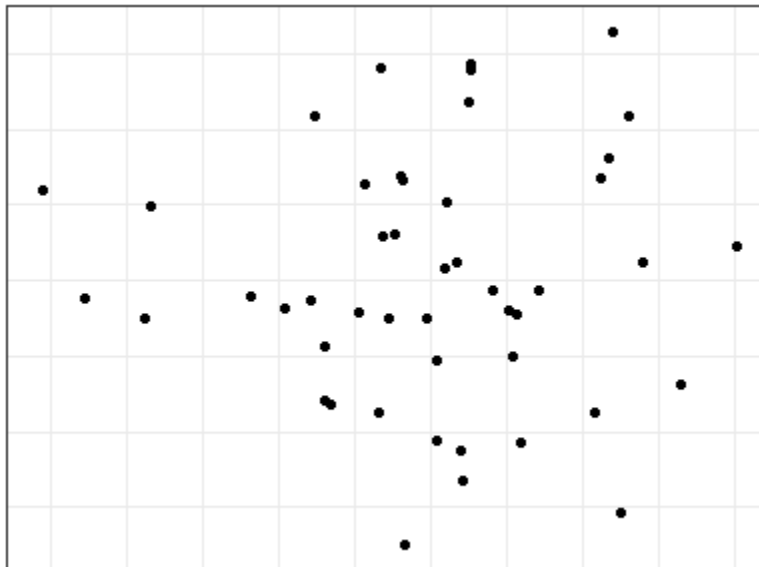
```
z.points <- data.frame(z$points)
```

Using ggplot2

Once we have the coordinates of the points, we can graph the ordination using `ggplot2`. In this case, I'm going to save the formatting aspects of this code as its own object so that I can easily re-use it below.

```
p <- ggplot(data = z.points, aes(x = MDS1, y = MDS2)) +
  theme_bw() +
  theme(axis.title = element_blank(),
        axis.ticks = element_blank(),
        axis.text = element_blank())

p + geom_point()
```



First two dimensions of a 3-dimensional NMDS ordination of the oak plant community dataset (47 stands). Each point is a stand. Stands near one another are similar in composition while those far apart in this space are different in composition. Stress = 0.16.

General Notes about Visualizations

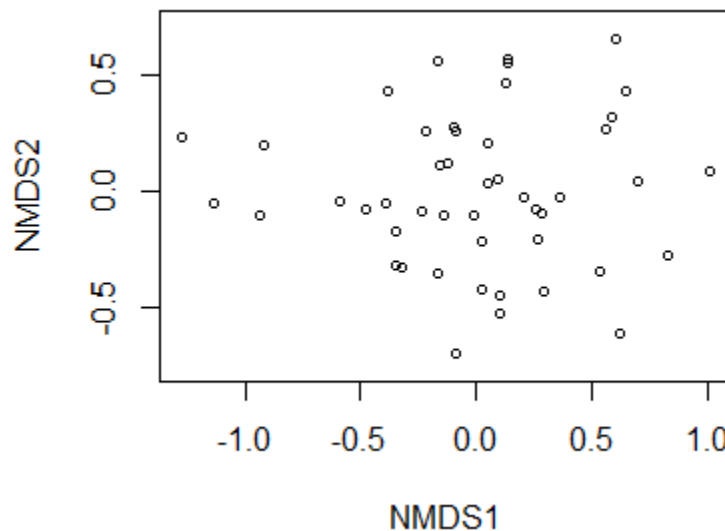
Recall that axes were rotated via PCA as the last step in the NMDS algorithm in `metaMDS()`. As a result, the axes are arranged so that they explain declining amounts of variation. We graphed the first two axes above, but it is important to remember that the axes in a NMDS do not have an inherent meaning like they do in PCA. It therefore is perfectly acceptable to look at the NMDS data cloud from other perspectives. This can take several forms:

- Viewing other combinations of axes. Obviously, this is only possible if you have specified a solution with more than two axes! In `ggplot2`, this is specified by changing `x` and `y`.
- When comparing an ordination to an explanatory variable: rotate an ordination so that the first axis is parallel to the explanatory variable. This can be done using the `MDSrotate()` function, which we'll discuss **later**.
- When comparing two ordinations based on data from the same sample units: orient them so that sample units from the same group are in the same relative location. Procrustes analysis is one method to do so.

Graphing with Base R Capabilities

We can use the `plot()` function to see a graph of the final configuration of the `metaMDS` results. By default, this function displays both sites and species. We can use the `display` argument to specify which of these we want. To display only sites:

```
plot(z, display = "sites")
```

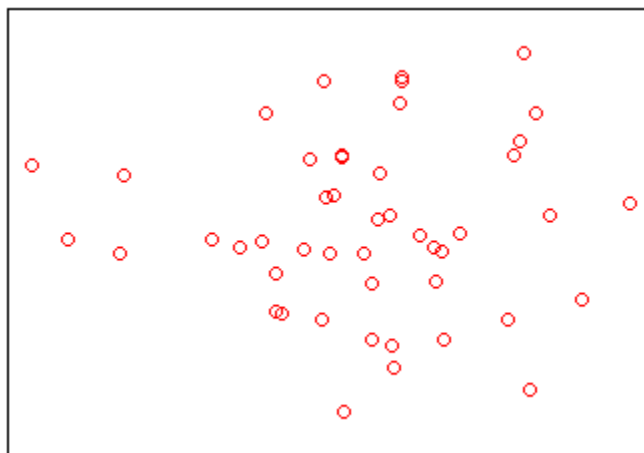


First two dimensions of a 3-dimensional NMDS ordination of the oak plant community dataset (47 stands). Each point is a stand. Stands near one another are similar in composition while those far apart in this space are different in composition. Stress = 0.16.

An alternative is to specify, while plotting, whether to plot points (`type = "p"`; the default) or text (`type = "t"`) or neither (`type = "n"`; a blank graph).

Creating a blank graph is helpful if we want to customize it. We begin by creating a plot of `z` but not displaying any points and hiding the axis labels and units. The net result is that the axes are scaled to fit the data even though they aren't shown. Then, we add the desired data as `points()`, `text()`, etc.

```
plot(z, display = "sites", type = "n",  
     xaxt = "n", xlab = "", yaxt = "n", ylab = "")  
  
points(z, col = "red")
```



First two dimensions of a 3-dimensional NMDS ordination of the oak plant community dataset (47 stands). Each point is a stand. Stands near one another are similar in composition while those far apart in this space are different in composition. Stress = 0.16.

Species Scores

Calculations

NMDS coordinates show the locations of the sample units relative to one another, but what about the species?

One reason some people like ordination approaches such as correspondence analysis (CA) is that the mathematics result in coordinates for both the sample units and the species in the same ordination space. In CA, each species is plotted at the peak of its assumed unimodal response curve.

While we cannot do the same in NMDS, we can approximate the location at which each species is most abundant. To do so for a given species, we weight the coordinates of all plots by the abundance of that species in the plots. We can do so with the `wascores()` function:

```
sp <- wascores(x = z$points, w = Oak1, expand = TRUE)
```

The `expand` argument scales the species scores so that they have the same weighted variance as the scores of the sample units. This function could also have been called using the `wascores` argument in the post-NMDS step of `metaMDS()`.

Personally, I have not found these species scores to be very useful. Knowing the weighted average

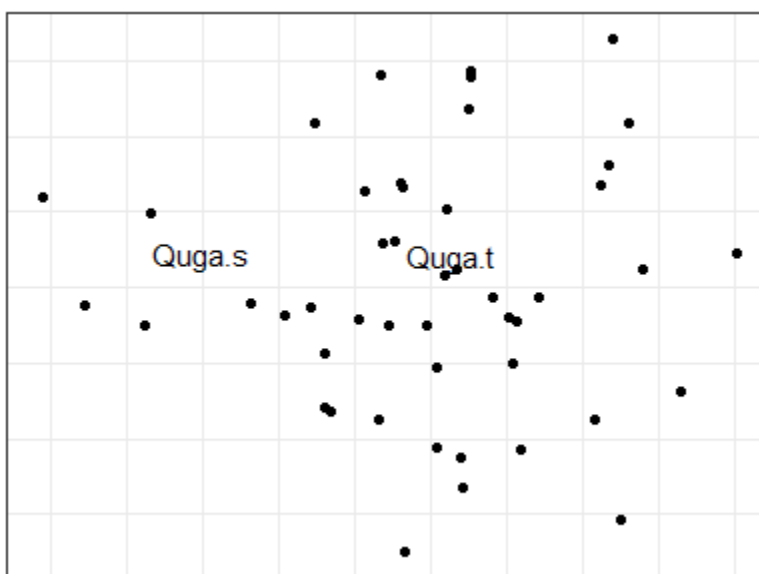
score of each species doesn't tell us, for example, whether a species is widespread or narrowly distributed. In my opinion, there are better ways to explore species. I illustrate some graphical techniques here, and also refer you to techniques such as Indicator Species Analysis and TITAN.

Graphing via ggplot2

Viewing all species at once is too busy to be interpretable (you're welcome to try it out!). Let's focus on two taxa, the shrubby and tree forms of Garry oak (*Quga.s* and *Quga.t*, respectively).

```
Quga <- data.frame(sp[c("Quga.s", "Quga.t"), ])
```

```
p +  
  geom_point() +  
  geom_text(data = Quga, label = rownames(Quga))
```



*First two dimensions of a 3-dimensional NMDS ordination of the oak plant community dataset (47 stands). Each point is a stand. Stands near one another are similar in composition while those far apart in this space are different in composition. Stress = 0.16. The text labels indicate where *Quga.s* and *Quga.t* are most abundant.*

Note that we have added these to the graphic *p* that we created earlier. Do you agree that these two centroids aren't too informative by themselves?

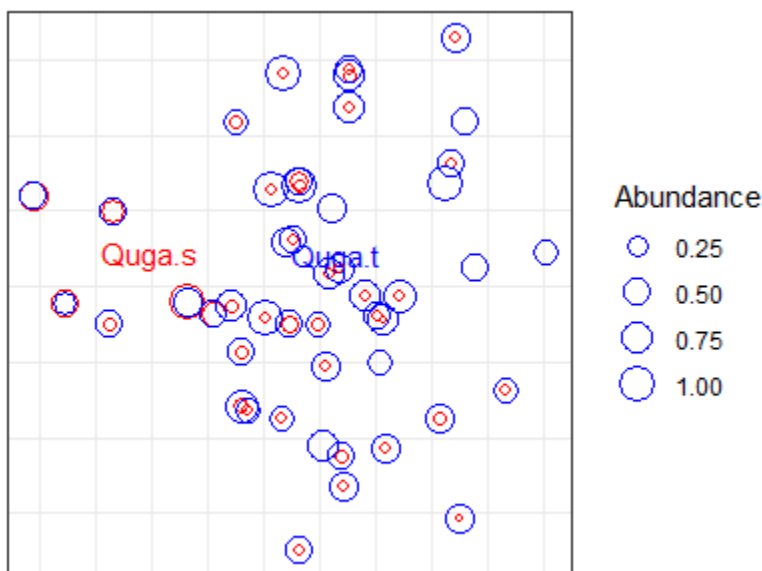
Let's incorporate the actual data. We'll make point sizes proportional to the abundance of *Quga.s* and *Quga.t*.

We begin by combining the NMDS coordinates with the abundance data for these species:

```
z.points <- data.frame(z.points,  
  Quga.s = Oak1$Quga.s,  
  Quga.t = Oak1$Quga.t)
```

Now we can use these abundance values in our graphing.

```
p +  
  geom_text(data = Quga, label = rownames(Quga),  
    colour = c("red", "blue")) +  
  geom_point(data = z.points[z.points$Quga.s > 0,],  
    aes(size = Quga.s), shape = 21, colour = "red") +  
  geom_point(data = z.points[z.points$Quga.t > 0,],  
    aes(size = Quga.t), shape = 21, colour = "blue") +  
  labs(size = "Abundance")
```



First two dimensions of a 3-dimensional NMDS ordination of the oak plant community dataset (47 stands). Each point is a stand. Stands near one another are similar in composition while those far apart in this space are different in composition. Stress = 0.16. Symbol color differs among species (red for Quga.s, blue for Quga.t) and symbol size reflects the relative abundance of each species in each stand. Stands have two symbols if both species were present.

While creating this graph, I added the species scores (i.e., centroids) as text. I then plotted two sets of points, one for each species. For each set of points, I:

- restrict the data to non-zero values for that species
- made symbol size proportional to the relative abundance of the species
- used a different color

Issues in NMDS

How Many Dimensions to Use?

One of the key differences between NMDS and the other ordination techniques we've covered is that **NMDS solutions are dependent on the number of dimensions selected**. A 2-D solution differs from the first two dimensions of a 3-D solution. The number of dimensions is often selected by conducting NMDS ordinations with different numbers of dimensions (e.g., $k = 1$ to 5) and then plotting stress as a function of k . The intent here is to identify the point where adding the complexity of an additional dimension contributes relatively little to the reduction in stress.

How many dimensions are appropriate? The 'correct' answer is a balance between interpretive issues (interpretability of axes, ease of use, stability of solution) and statistical issues (stress):

- Too many axes defeats the purpose of visualizing and interpreting the dataset
- Too few axes can result in unacceptably large stress values

This subjectivity in terms of the number of dimensions is important to keep in mind because, as noted above, the axes of a NMDS ordination do not have the same interpretation as those in a PCA or CCA. Instead, the focus of NMDS is on the data points themselves, and particularly on their orderings relative to one another. As Clarke (1993) described it: "sample A is more similar to sample B than it is to sample C" (p. 117). For this reason, and as stated earlier, **NMDS ordinations are often plotted without units or labels on the axes so that people do not try to interpret them**.

The stress of a solution always decreases as the number of dimensions increases – adding another dimension means that there is more 'room' for adjustments in the coordinate space to increase the alignment between those distances and the original distance matrix.

Specifying more dimensions in a solution does not mean that you have to show them all graphically. For example, above we did a 3-dimensional NMDS but only displayed the first two dimensions. Since the coordinates were rotated via PCA, these dimensions show as much of the variation in the coordinates as possible in two dimensions. However, note again that the first two dimensions of a 3-dimensional solution are not the same as a 2-dimensional solution.

The significance of a solution can be assessed using Monte Carlo techniques to determine the likelihood of the observed stress values. To my knowledge, these techniques are not currently available in R (though it would be relatively easy to write a function that would perform these steps). The computational limitations of this technique have diminished greatly since McCune & Grace (2002) was published. Therefore, it is feasible to conduct more Monte Carlo tests than they recommend.

Unstable Solutions

Stress can be graphed as a function of the iteration number to assess fluctuations, stability, etc. This was shown in the YouTube video referenced earlier (see 'NMDS in Action' section), but this type of graph is not available in the current NMDS functions in R. I have not yet found a way to track how stress changes with iteration number. The closest I am aware of is that the stress of the solution based on each random start is displayed on the screen (the `trace = 1` argument in `metaMDS()`) but is not saved. The `metaMDS()` function could be altered to save this information...

Relation to Explanatory Variables

NMDS ordinations are based solely on a response matrix. We are often naturally very interested in comparing the resulting patterns to explanatory variables, both categorical (e.g., burned or not) and continuous (e.g., elevation). We can explore these relationships qualitatively by overlaying them onto the ordination – and will do so **later**. However, it is important to keep in mind when doing so that these patterns may be affected by how well the ordination fits the distance matrix. **When overlaying explanatory variables onto an NMDS ordination, we are visualizing their relationship with the reduced dimensionality of the ordination space, not with the original distance matrix.** In contrast, analytical techniques such as PERMANOVA quantify the relationship between explanatory variables and the distance matrix derived directly from the data.

Strengths and Interpretive Difficulties with NMDS

Strengths of NMDS:

- Avoids assumptions about form of relationships among variables
- Uses ranked distances to linearize relationships
- Allows use of any transformation, standardization, and distance measure. This allows the analytical methods to be determined by the ecology of the dataset. It also means that the ordination, formal tests of statistical significance (e.g., PERMANOVA, ANOSIM), and/or classification techniques (e.g., cluster analysis) can all use data that has received the same adjustments and thus are directly comparable with one another.

Interpretive Difficulties of NMDS:

- May fail to find the global solution (minimum stress)
- Slow computation with large datasets
- Interpretation is dependent on dimensionality of the ordination – which has to be specified by the user
- Based on the rank order of the distances, not on the distances themselves

Conclusions

Minchin (1987) compared PCA, PCoA, DCA, and NMDS, and concluded that NMDS is “a robust technique for indirect gradient analysis, which deserves more widespread use by community ecologists” (p. 89). McCune & Grace (2002) felt that NMDS and related techniques are the way ordination will be increasingly conducted in ecology. They noted that NMDS “is currently one of the most defensible techniques during peer review” (p. 125). von Wehrden et al. (2009) found that its usage increased between 1990 and 2007.

The relative differences among some of these models are unclear; Anderson (2015) notes that techniques like NMDS and PCoA will tend to give similar results, and that far more important differences arise from decisions about data adjustments (transformations, standardizations) and which distance measure to use.

References

- Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth, UK.
- Anderson, M.J. 2015. *Workshop on multivariate analysis of complex experimental designs using the PERMANOVA+ add-on to PRIMER v7*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK.
- Anderson, M.J., and T.J. Willis. 2003. Canonical analysis of principal coordinates: a useful method of constrained ordination for ecology. *Ecology* 84:511-525.
- Borcard, D., F. Gillet, and P. Legendre. 2011. *Numerical ecology with R*. Springer, New York, NY.
- Clarke, K.R. 1993. Non-parametric multivariate analyses of changes in community structure. *Australian Journal of Ecology* 18:117-143.
- De'ath, G. 1999. Extended dissimilarity: a method of robust estimation of ecological distances from high beta diversity data. *Plant Ecology* 144:191-199.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.
- Gower, J.C. 1966. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika* 53:325-338.
- Haugo, R.D., C.B. Halpern, and J.D. Bakker. 2011. Landscape context and tree influences shape the long-term dynamics of forest-meadow ecotones in the central Cascade Range, Oregon. *Ecosphere* 2:art91.
- Kruskal, J.B. 1964a. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29:1-27.
- Kruskal, J.B. 1964b. Nonmetric multidimensional scaling: a numerical method. *Psychometrika* 29:115-129.
- Laliberté, E., and P. Legendre. 2010. A distance-based framework for measuring functional diversity from multiple traits. *Ecology* 91:299-305.
- Legendre, P., and M.J. Anderson. 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs* 69:1-24.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McArdle, B.H. and M.J. Anderson. 2001. Fitting multivariate models to community data: a comment on distance-based redundancy analysis. *Ecology* 82:290-297.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Minchin, P.R. 1987. An evaluation of the relative robustness of techniques for ecological ordination. *Vegetatio* 69:89-107.
- Shepard, R.N. 1962a. The analysis of proximities: multidimensional scaling with an unknown distance function. I. *Psychometrika* 27:125-140.
- Shepard, R.N. 1962b. The analysis of proximities: multidimensional scaling with an unknown distance function. II. *Psychometrika* 27:219-246.
- von Wehrden, H., J. Hanspach, H. Bruelheide, and K. Wesche. 2009. Pluralism and diversity: trends in the use and application of ordination methods 1990-2007. *Journal of Vegetation Science* 20:695-705.

Media Attributions

- NMDS.demo1
- NMDS.demo2
- goodness
- shepard
- stressplot
- NMDS.plain
- NMDS.base
- NMDS.base.red
- NMDS.Quga.plain
- NMDS.Quga

38. PCoA

Learning Objectives

To understand how PCoA works and when it is an appropriate ordination technique to use.

Key Packages

```
require(vegan, tidyverse)
```

Introduction

Principal Coordinates Analysis (PCoA) is an unconstrained or indirect gradient analysis ordination method. It was first published by Gower (1966). The appealing element of PCoA is that it can be applied to semi-metric distance measures – those that do not satisfy the triangle inequality. If it is applied to a metric distance measure, it functionally is identical to a PCA.

Confusingly, PCoA is also abbreviated PCO, and is also known as metric MultiDimensional Scaling (MDS). Some authors distinguish PCoA from metric MultiDimensional Scaling (MDS) based on their numerical approach. For example, Manly & Navarro Alberto (2017) state that PCoA is an eigenanalysis technique while MDS attempts to minimize stress. However, others equate them – for example, the help files for `cmdscale()` state that the function performs classical multidimensional scaling (MDS), aka PCoA. We won't worry about these distinctions here.

For an example of how PCoA has been applied in an unusual context, see Metcalf et al. (2016)!

How PCoA Works

In a blog post (https://occamstypewriter.org/boboh/2012/01/17/pca_and_pcoa_explained/), Bob O'H provides a verbal description of the PCoA process, which I summarize here.

The goal of PCoA is to find a set of Euclidean distances that represent a set of non-Euclidean distances. Borcard et al. (2018) describe PCoA as providing “a Euclidean representation of a set of objects whose relationships are measured by any dissimilarity measure chosen by the user” (p. 187).

If you only have two sample units, there is one distance to consider and therefore you can illustrate that distance on a (one-dimensional) number line. If you have three sample units, there are also three distances to consider and illustrating these patterns will require one or two dimensions – either three points along a straight line or three points on a plane. The general principle that this reflects is that n sample units can be illustrated or mapped in no more than $n-1$ dimensions.

With this context, let's summarize the PCoA procedure:

1. Place the first point at the origin.
2. Add the second point the correct distance away from the first point along the first axis.
3. Position the third point the correct distance away from each of the first two points, adding a second axis if necessary.
4. Position the fourth point the correct distance away from each of the first three points, adding another axis if necessary.
5. Continue until all points have been added. The result is a set of no more than $n-1$ axes.
6. Conduct a PCA on the constructed points to organize the variation among the points in a series of axes of diminishing importance.

PCoA is intended for use with non-Euclidean distances and dissimilarities. When applied to semi-metric distance measures (i.e., those that do not satisfy the triangle inequality), some of the resulting axes may be in 'imaginary space' (Anderson 2015). These axes are mathematically possible but difficult to envision – you can't graph them, for example. When they are subject to PCA (see last step above), these imaginary axes produce negative eigenvalues.

Two types of adjustments are possible that adjust the distance matrix to avoid negative eigenvalues:

- Lingoes correction – add a constant to the squared dissimilarities
- Cailliez correction – add a constant to the dissimilarities (Cailliez 1983)

Legendre & Anderson (1999) and Borcard et al. (2018) recommend using the Lingoes correction method but others use the Cailliez correction. For example, in the RRPP chapter we noted that `adonis2()` and `lm.rrpp()` did not give the same results when applied to a Bray-Curtis distance matrix, and that this was because `lm.rrpp()` automatically makes this adjustment while `adonis2()` does not. The author of the RRPP package noted that the outputs of the two functions agree if `adonis2()` uses the Cailliez correction method. Making either of these adjustments has implications that I have not seen fully explored – see [Appendix](#) for one example.

Aside: if PCoA is applied to a set of Euclidean distances, it produces the same set of distances as in the original set and therefore does not change the distances.

Additional details about PCoA, and extensions to it, are provided in Anderson et al. (2008, ch. 3) and Borcard et al. (2018, ch. 5.5).

Similarities to PCA and NMDS

PCoA has similarities to both PCA and NMDS.

Like PCA...

Like PCA, PCoA is an eigenanalysis technique. Manly & Navarro Alberto (2017; ch. 12.3) provide a detailed illustration of the connection between PCA and PCoA. The fact that they are both eigenanalysis-based techniques means that:

- Points are projected onto axes.
- Since a PCA is the last step of a PCoA, the first axis necessarily explains as much of the variation as possible, the second axis explains as much of the remaining variation as possible, etc.
- As with PCA, PCoA is rotating and translating the data cloud. If all axes are retained, no data reduction has occurred and the relative locations of the points to one another are unchanged. This is true even when applied to non-Euclidean distances.
- The 'fit' of the PCoA can be assessed by the percentage of variation explained by each axis.

A PCoA using Euclidean distances is identical to a PCA of the same data. Similarly, a PCoA using chi-square distances is identical to a CA of the same data (Anderson 2015). In other words, **PCA and CA are variants of PCoA, which is the more general technique.**

Like NMDS...

Like NMDS, PCoA is a distance-based ordination technique and thus can be applied to any distance measure. However, **PCoA maximizes the agreement between the actual distances in ordination space and in the original space**, whereas NMDS seeks only to maintain the rank order of the distances in the two matrices. The help files for `cmdscale()` describes PCoA as returning "a set of points such that the distances between the points are approximately equal to the dissimilarities". Because of this focus on linear relationships, it has been suggested that PCoA is more appropriate than NMDS as an accompaniment to a PERMANOVA model (Anderson 2015).

PCoA in R (`vegan::wcmdscale()`)

In R, PCoA can be accomplished using `stats::cmdscale()`, `vegan::wcmdscale()`, and `ape::pcoa()`. I'm sure other functions also exist for this purpose.

The `wcmdscale()` function is a wrapper that calls `cmdscale()` while providing some additional capabilities. Its usage is:

```
wcmdscale(d,
  k,
  eig = FALSE,
  add = FALSE,
  x.ret = FALSE,
  w
)
```

The arguments are:

- `d` – distance matrix to be analyzed.
- `k` – number of dimensions to save. Default is to save all non-negative eigenvalues (dimensions). Up to $n-1$ eigenvalues are possible.
- `eig` – whether to save the eigenvalues in the resulting object. Default is to not do so (`FALSE`).
- `add` – whether to add a constant to the dissimilarities so that all eigenvalues are non-negative. Default is to not do this (`FALSE`) but two options are possible:
 - `"lingoes"` or `TRUE` – this is the option recommended by Legendre & Anderson (1999) and Borcard et al. (2018).
 - `"cailliez"` – this option is required for `adonis2()` results to match those from `lm.rppp()`.

- `x.ret` – whether to save the distance matrix. Default is to not do so.
- `w` – weights of points (sample units). If set to equal weights (`w = 1`; the default), this gives a regular PCoA. If sample units are given different weights, those with high weights will have a stronger influence on the result than those with low weights.

The output of this function depends on the arguments selected. If `eig = FALSE` and `x.ret = FALSE`, this returns just a matrix of coordinates for each positive eigenvalue. Otherwise, this returns an object of class 'wcmdscale' with several components, including:

- `points` – the coordinates of each sample unit in each dimension (`k`).
- `eig` – the eigenvalue of every dimension. As noted above, some may be negative.
- `ac` – additive constant used to avoid negative eigenvalues. NA if no adjustment was made.
- `add` – the adjustment method used to avoid negative eigenvalues. FALSE if no adjustment was made.

An object of class 'wcmdscale' can be subject to pre-defined methods for `print()`, `plot()`, `scores()`, `eigenvals()`, and `stressplot()`.

Oak Example

Use the 'load.oak.data.R' script to load our oak data and make our standard adjustments to it.

```
source("scripts/load.oak.data.R")
```

Now, apply a PCoA to our oak plant community dataset:

```
Oak1_PCoA <- wcmdscale(d = Oak1.dist, eig = TRUE)
```

Explore the output. Try changing the `k` and `eig` arguments and see how the output changes (hint: `str()`).

```
print(Oak1_PCoA)
```

```
Call: wcmdscale(d = Oak1.dist, eig = TRUE)
```

	Inertia	Rank
Total	11.5949	
Real	12.0707	34
Imaginary	-0.4758	12

```
Results have 47 points, 34 axes
```

```
Eigenvalues:
```

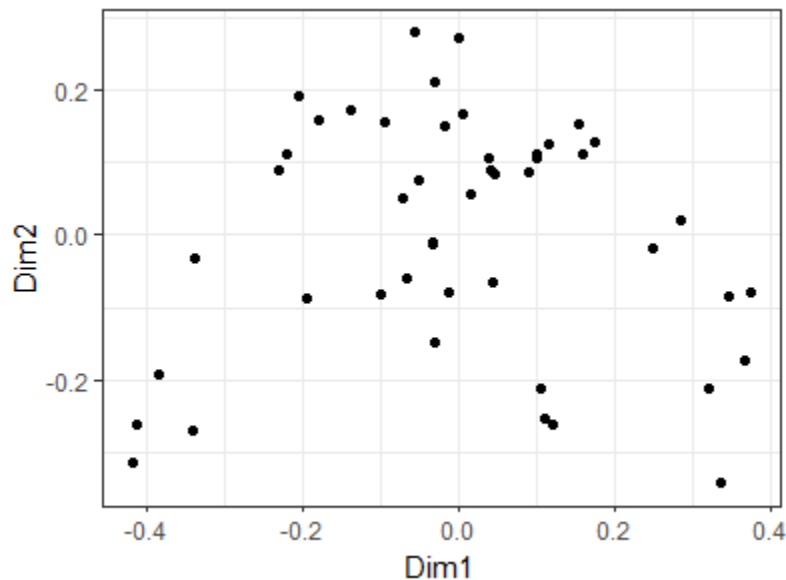
```
[1] 1.9133 1.2359 1.1947 0.8806 0.7649 0.6919 0.5303 0.4490
[9] 0.4158 0.4001 0.3593 0.3388 0.3192 0.2857 0.2747 0.2413
[17] 0.2236 0.2012 0.1832 0.1670 0.1517 0.1384 0.1288 0.1112
[25] 0.1032 0.0880 0.0788 0.0552 0.0498 0.0354 0.0282 0.0157
[33] 0.0143 0.0014 -0.0021 -0.0102 -0.0124 -0.0240 -0.0295 -0.0334
[41] -0.0362 -0.0426 -0.0593 -0.0613 -0.0740 -0.0909
```

Weights: Constant

Note the presence of negative eigenvalues – in this case for 12 dimensions. These are the additional dimensions that were required to fit the semi-metric Bray-Curtis distances into a Euclidean space. However, the last step of the PCoA process was to conduct a PCA -this means that the coordinates have been rotated such that the axes are in descending order of importance. Therefore, these dimensions account for small amounts of variation in the dataset. If your objective is to reduce the dimensionality of the data, these dimensions can be ignored without changing your conclusions greatly. Try changing the `add` argument and see how these are affected.

The output is a list as for `metaMDS()`, though this one isn't as detailed. The coordinates of the sample units are again reported as the `points` element within the list, though the axes are named differently than for a NMDS. We'll plot the first two axes using `ggplot2`.

```
ggplot(data = data.frame(Oak1_PCoA$points),
  aes(x = Dim1, y = Dim2)) +
  geom_point() +
  theme_bw()
```



I displayed the axis coordinates here, but we could omit them to force the viewer to focus on the data cloud. See the 'Graphing the Final Configuration' section from the NMDS chapter for an example of how to do so.

Uses of PCoA

We have already encountered PCoA several times:

- Displaying results of a PERMDISP analysis.
- Default option for generating the starting configuration for a NMDS.

Examples of other ways PCoA can be used:

- Combining functional traits of different organisms. The FD package contains a `dbFD()` function that combines traits in different ways depending on their class and then combines the traits using PCoA (Laliberté & Legendre 2010).
- Expressing the location of centroids. This is helpful, for example, when composition has been measured in multiple quadrats (sub-samples) per plot but where plots are the experimental units. We could conduct a PCoA using the quadrats, keeping all dimensions. Since these distances are now expressed in a Euclidean space, even if the original distance matrix was not, we can then calculate the centroid for each plot based on the locations of the quadrats in this ordination space. The coordinates of these plots can then be the subject of statistical analysis. An advantage of this approach is that it avoids the requirement of a balanced dataset for restricted permutations. For example, you could calculate the centroid for each plot based on however many quadrats were sampled in that plot.

Interpreting PCoA

Since PCoA seeks to maximize the agreement between the actual distances in ordination space and in the original space, it is more similar to PERMANOVA than NMDS, which is based on the ranks of the distances. Conceptually, therefore, the PCoA approach is more consistent with PERMANOVA, though I am not aware of any strong tests of this. In addition, Anderson (2015) notes that NMDS generally works better than PCoA when visualizing a multi-dimensional distance matrix in a few dimensions.

As with PCA, keeping all axes means that the data cloud has been rotated but that its dimensionality has not been reduced. This can be helpful, for example, because it allows you to express a non-Euclidean distance matrix in Euclidean units. See the above section about 'Uses of PCoA' for examples.

Some people 'object' to the negative eigenvalues. These are only an issue for distance measures that do not satisfy the triangle inequality. Optional settings can be used to adjust the values so that no negative eigenvalues are produced (but see the [Appendix](#) for an illustration of how this adjustment can affect interpretation of results). Even if these adjustments are not used, remember that if your purpose is data reduction, you are most interested in the first few dimensions – and those dimensions are rarely negative.

Conclusions

Several of these techniques are closely related. For example, PCA and CA are applications of PCoA to particular types of distance measures.

The relative differences among some of these techniques are unclear; Anderson (2015) notes that techniques like NMDS and PCoA will all tend to give very similar results, and that far more important

differences arise from decisions about data adjustments (transformations, standardizations) and which distance measure to use.

References

Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth, UK.

Anderson, M.J. 2015. *Workshop on multivariate analysis of complex experimental designs using the PERMANOVA+ add-on to PRIMER v7*. PRIMER-E Ltd, Plymouth Marine Laboratory, Plymouth, UK.

Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.

Cailliez, F. 1983. The analytical solution of the additive constant problem. *Psychometrika* 48:305-308.

Gower, J.C. 1966. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika* 53:325-338.

Laliberté, E., and P. Legendre. 2010. A distance-based framework for measuring functional diversity from multiple traits. *Ecology* 91:299-305.

Legendre, P., and M.J. Anderson. 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs* 69:1-24.

Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.

Metcalfe, J.L., Z.Z. Xu, S. Weiss, S. Lax, W. Van Treuren, E.R. Hyde, S.J. Song, A. Amir, P. Larsen, N. Sangwan, D. Haarmann, G.C. Humphrey, G. Ackermann, L.R. Thompson, C. Lauber, A. Bibat, C. Nicholas, M.J. Gebert, J.F. Petrosino, S.C. Reed, J.A. Gilbert, A.M. Lynne, S.R. Bucheli, D.O. Carter, and R. Knight. 2016. Microbial community assembly and metabolic function during mammalian corpse decomposition. *Science* 351(6269):158-162.

Media Attributions

- PCoA_gg

39. RDA and dbRDA

Learning Objectives

To consider when and how to conduct a constrained ordination.

Readings

Type your exercises here.

Key Packages

```
require(vegan, tidyverse)
```

Contents:

- Introduction
- ReDundancy Analysis (RDA)
- Distance-based ReDundancy Analysis (dbRDA)
- Oak Example
- Interpreting RDA and dbRDA
- Conclusions
- References

Introduction

Other than CCA, all of the ordination techniques that we've discussed thus far are unconstrained or indirect gradient analyses – the only information that is incorporated into the procedure is the matrix of response variables. **Constrained** ordinations are **direct gradient analyses**. These are analogous to regression, where the objective is to correlate the response matrix and one or more explanatory variables.

A 3-D conceptual example might help here. Imagine the sample units as a point cloud. We could view this point cloud from many different vantage points. For example, we could view it from a spot where we see as much of the variation as possible – an unconstrained ordination is showing us the view from this spot. Now imagine that the points are colored to reflect different groups. Depending on where we view the point cloud from, these groups might appear more or less discrete. A constrained ordination shows us the view from the spot where the groups are as distinct as possible. These different perspectives are illustrated in the [appendix](#).

In summary, different types of analyses provide different ways to view the point cloud:

- Unconstrained ordinations identify the perspective that shows as much of the variation in the point cloud as possible.
- Constrained ordinations view or interpret the point cloud from the perspective of the chosen explanatory variable(s).

We will consider two similar types of constrained ordinations, ReDundancy Analysis (RDA) and distance-based RDA (dbRDA). Neither of these techniques is addressed by Manly & Navarro Alberto (2017).

ReDundancy Analysis (RDA)

Approach

Legendre & Legendre (2012; Section 11.1) describe ReDundancy Analysis (RDA) as “the direct extension of multiple regression to the modelling of multivariate response data” (p. 629). It is based on Euclidean distances. Transformations and standardizations can be conducted at the user’s discretion outside of the analysis.

RDA uses two common techniques in sequence: “RDA is a multivariate (meaning multiresponse) multiple linear regression followed by a PCA of the matrix of fitted values” (Borcard et al. 2018, p. 205).

1. **Regression:** Each variable in the response matrix is fit to the variables in the explanatory matrix. The regression model assumes that the explanatory and response variables are linearly related to one another. This may require transformations. The fitted values of each response variable with each explanatory variable are calculated. The residuals (i.e., variation not accounted for by the explanatory variables) are also calculated – these are simply the original values minus the fitted values.
2. **PCA:** Two separate PCAs are conducted:
 1. A PCA on the matrix of fitted values. By conducting the PCA on the fitted values rather than the original values, the resulting eigenvalues and eigenvectors are constrained to be linear combinations of the explanatory variables.
 2. A PCA on the matrix of residuals. These residuals are conditional: they are the variation that was not explained by the explanatory variables.

Notes:

- If no explanatory variables are provided, then the first step is meaningless: there are no fitted values, and the second step results in a regular PCA.
- The explanatory variables should not be highly correlated with one another. If two such variables are highly correlated, consider omitting one or using PCA first to reduce them to a

single principal component.

The result of this process is a set of principal components conditioned (or weighted) by the explanatory variables. “In RDA, one can truly say that the axes *explain* or *model* (in the statistical sense) the variation of the dependent [response] matrix” (Borcard et al. 2018, p. 205).

The degrees of freedom (df) required for the explanatory variables indicate how many axes are constrained by those variables. For example, a single continuous variable requires 1 df, and a factor with m levels requires $m-1$ df. The axes that are constrained by these variables are referred to as **canonical axes** or **constrained axes**. As with a regular PCA, each of these axes is orthogonal to the others, and can be tested and evaluated individually. Borcard et al. (2018) note that the eigenvalues associated with these axes measure the amount of variance explained by the RDA model.

Once the constrained axes have accounted for all of the variation that they can, the remaining variation is arrayed along **unconstrained axes**. These are interpreted in the same way as in a PCA, except that they are explaining the variation that remains after having accounted for as much variation as possible through the constrained axes. Borcard et al. (2018) note that the eigenvalues associated with these axes measure the amount of variance represented by these axes, but not explained by any included explanatory variable.

I have noted several times that PCs are always a declining function: the first explains as much variation as it can, the second as much as it can of the remaining, etc. Constrained ordinations are a partial exception to this rule. The constrained axes are a declining function and the unconstrained axes are also a declining function, but the transition from the last constrained axis to the first unconstrained axis may be associated with a (sometimes dramatic) increase in the magnitude of the eigenvalue.

RDA in R (**vegan::rda()**)

In R, RDA can be conducted using function `rda()` in **vegan**. Its usage is:

```
rda(formula,
    data,
    scale = FALSE,
    na.action = na.fail,
    subset = NULL,
    ...
)
```

The arguments are:

- **formula** – model formula, with the community data matrix on the left hand side and the explanatory (constraining) variables on the right hand side. Additional explanatory variables can be included that ‘condition’ the response – their effect is removed before testing the constraining variables. See the help files for details of how to include conditioning variables. The explanatory and response variables can also be specified as separate arguments, but the formula approach is preferred as it allows you to easily control which variables are being used.
- **data** – the data frame containing the explanatory variables.
- **scale** – whether to scale species (i.e., columns in the response matrix) to a variance of 1 (like a correlation). Default is to not do so (**FALSE**). While this may make sense with other types of responses, Borcard et al. (2018) state that this should not be used with community data.
- **na.action** – what to do if there are missing values in the explanatory variables. Several options:
 - **na.fail** – stop analysis. The default.
 - **na.omit** – drop rows with missing values and then proceed with analysis.

- `na.exclude` – keep all observations but return `NA` for values that cannot be calculated.
- `subset` – whether to subset the data, and how to do so.

The resulting object contains lots of different elements. An object of class 'rda' can be subject to pre-defined methods for `plot()`, `print()`, and `summary()`.

We have talked repeatedly about how it is more appropriate to summarize community composition data using Bray-Curtis rather than Euclidean distances. I therefore will not illustrate RDA using our oak plant community dataset but will instead illustrate dbRDA below.

distance-based ReDundancy Analysis (dbRDA)

Approach

ReDundancy Analysis (RDA) is a constrained ordination that uses Euclidean distances. Distance-based ReDundancy Analysis (dbRDA or db-RDA; Legendre & Anderson 1999) is a more general way to do the same type of constrained ordination; the generality arises because it can be applied to any distance or dissimilarity matrix.

The connection between RDA and dbRDA is simply the step of expressing a non-Euclidean dissimilarity matrix in a Euclidean space. Sound familiar? That's what a PCoA does.

As a result, the first step of a dbRDA is to apply a PCoA to the distance matrix and to keep all principal coordinates. As discussed in that chapter, this can be done regardless of the distance measure used (hence the generality of this technique). By keeping all of the principal coordinates we have rotated the distance matrix and expressed it in Euclidean distances without reducing its dimensionality.

The second step of a dbRDA is to conduct a RDA on the principal coordinates. In other words, the principal coordinates are subject to a multivariate multiple linear regression and then the fitted values and residuals are subject to separate PCAs. This procedure maximizes the fit between the distance matrix containing the PCoA coordinates and the explanatory variables or constraints.

Notes:

1. If applied to a Euclidean distance matrix, dbRDA is identical to RDA.
2. If no explanatory variables are provided, a dbRDA is identical to a PCoA (because the first step results in a regular PCoA and the second step is meaningless).
3. Canonical Analysis of Principal coordinates (CAP) is another technique that has been proposed (McArdle & Anderson 2001; Anderson & Willis 2003). According to Borcard et al. (2018), it analyzes the dissimilarity matrix directly and does not require the initial PCoA. However, the help file for `vegan::capscale()` indicates that it is similar to dbRDA, and it appears that it is not maintained as a separate function.

db-RDA in R (`vegan::dbrda()` and `vegan::capscale()`)

The help file describes these functions as 'alternative implementations of dbRDA'. These functions differ somewhat in their approach – including how they deal with and report on negative

eigenvalues – but generally should give very similar results; see the help file for details. Both are wrapper functions that call on other functions for various steps in the analysis.

The usage of `dbrda()` is:

```
dbrda(formula,
      data,
      distance = "euclidean",
      sqrt.dist = FALSE,
      add = FALSE,
      dfun = vegdist,
      metaMDSdist = FALSE,
      na.action = na.fail,
      subset = NULL,
      ...
)
```

The key arguments are:

- `formula` – model formula, with a community data frame or distance matrix on the left hand side and the explanatory (constraining) variables on the right hand side. Additional explanatory variables can be included that condition the response – their effect is removed before testing the constraining variables. The explanatory and response variables can also be specified as separate arguments, but the formula approach is preferred as it allows you to easily control which variables are being used.
- `data` – the data frame containing the explanatory variables.
- `distance` – type of distance matrix to be calculated if the response is a data frame. Default is to use Euclidean distance.
- `sqrt.dist` – whether to transform the distance matrix by taking the square root of all dissimilarities. This can be helpful to ‘euclidify’ some distance measures. Default is to not do so (`FALSE`).
- `add` – when conducting the PCoA, whether to add a constant to the dissimilarities so that none of them are non-negative. Same options as for `wcmdscale()`: `FALSE` or `TRUE` or `"lingoes"` or `"cailliez"`. However, the default here is to not add a constant (`FALSE`) or, in other words, to permit negative eigenvalues in the results. Setting this to `TRUE` is the same as `add = "lingoes"`.
- `dfun` – name of function used to calculate distance matrix. Default is `vegdist` (i.e., `vegdist()`).
- `metaMDSdist` – whether to use the same initial steps as in `metaMDS()` (auto-transformation, stepacross for extended dissimilarities). Default is to not do so (`FALSE`).
- `na.action` – what to do if there are missing values in the constraints (explanatory variables). Same options as for `wcmdscale()`. Default is to stop (`na.fail`); other options are to drop sample units with missing values (`na.omit` or `na.exclude`).
- `subset` – whether to subset the data, and how to do so.

The resulting object contains lots of different elements. Objects of class ‘`dbrda`’ or ‘`capscale`’ can be subject to pre-defined methods for `plot()`, `print()`, and `summary()`. By default, only the first 6 axes are reported – since the objective is to view patterns related to one or more explanatory variables, the user is unlikely to need or use other axes.

The function `capscale()` has extremely similar usage – see the help file for details.

Oak Example

Let's use dbRDA to test the relationship between oak plant community composition and geographic variables.

```
source("scripts/load.oak.data.R")

geog.stand <- Oak %>%
  select(c(LatAppx, LongAppx, Elev.m)) %>%
  decostand("range")

Oak1_dbrda <- dbrda(Oak1 ~ Elev.m + LatAppx + LongAppx,
  data = geog.stand,
  distance = "bray")

print(Oak1_dbrda)
```

```
Call: dbrda(formula = Oak1 ~ Elev.m + LatAppx + LongAppx, data =
geog.stand, distance = "bray")
```

	Inertia	Proportion	Rank	RealDims
Total	11.59490	1.00000		
Constrained	0.97790	0.08434	3	3
Unconstrained	10.61700	0.91566	43	33

Inertia is squared Bray distance

Eigenvalues for constrained axes:

dbRDA1	dbRDA2	dbRDA3
0.4568	0.3043	0.2168

Eigenvalues for unconstrained axes:

MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
1.7901	1.1519	1.0933	0.7744	0.7069	0.6514	0.5212	0.4344

(Showing 8 of 43 unconstrained eigenvalues)

Much more extensive results is available through `summary()` and can be viewed in the Console or saved to an object:

```
Oak1_dbrda_summary <- summary(Oak1_dbrda)
```

Saving this output to an object allows its components to be explored and used for other purposes.

Interpreting the Output

Near the top of the output is a table showing of how the variance has been partitioned, which I've repeated here:

	Inertia	Proportion	Rank	RealDims
Total	11.59490	1.00000		

Constrained	0.97790	0.08434	3	3
Unconstrained	10.61700	0.91566	43	33

In this case, 8.43% of the variance is accounted for by the constraining variables, the set of geographic variables. The other 91.57% of the variance is unconstrained – it cannot be explained by the spatial location of the sample units.

Constrained Axes

The `summary()` output includes a summary of the constrained eigenvalues:

`Oak1_dbrda_summary$concont`

```
$importance
Importance of components:
              dbRDA1 dbRDA2 dbRDA3
Eigenvalue      0.4568 0.3043 0.2168
Proportion Explained 0.4671 0.3112 0.2217
Cumulative Proportion 0.4671 0.7783 1.0000
```

As with other eigenvalues, these are a declining function: the first always accounts for the most variation, the second for the second-most variation, etc. However, note that the proportion explained here is **not** the overall importance of each axis relative to the full variance in the dataset. Rather, it is the proportion of the variance explained by these explanatory variables. For example, dbRDA1 explains 46.71% of the 8.43% of the variance accounted for by these three variables. Multiplying those two values together yields 3.94%, which is the amount of variance explained by dbRDA1 overall.

The relative importance of axes can be viewed. By default, the first 6 dimensions are shown, even though only the first three axes are constrained in this example:

`Oak1_dbrda_summary$cont`

```
$importance
Importance of components:
              dbRDA1 dbRDA2 dbRDA3 MDS1 MDS2 MDS3
Eigenvalue      0.45675 0.30432 0.2168 1.7901 1.15188 1.09331
Proportion Explained 0.03939 0.02625 0.0187 0.1544 0.09934 0.09429
Cumulative Proportion      NA      NA      NA      NA      NA      NA
```

Since we included three explanatory variables, the first three axes (dbRDA1, dbRDA2, dbRDA3) are constrained to maximize the correlation between the response matrix and the geographic explanatory variables. These axes are, “in successive order, a series of linear combination of the explanatory variables that best explain the variation of the response matrix” (Borcard et al. 2018, p.155). In other words, each axis represents a multiple regression model based on all explanatory variables, that yield fitted values for the response.

The coefficient associated with each explanatory variable on each axis is reported in the 'biplot' component of the summary object:

```
Oak1_dbrda_summary$biplot
```

	dbRDA1	dbRDA2	dbRDA3	MDS1	MDS2	MDS3
Elev.m	-0.5518752	0.82653295	-0.1108018	0	0	0
LatAppx	0.7748542	0.46202768	0.4314295	0	0	0
LongAppx	0.6150552	-0.02390781	-0.7881215	0	0	0

Note that the explanatory variables have non-zero coefficients for the constrained axes but zeroes for the unconstrained axes. These coefficients are interpreted like the loadings from a PCA – they are the eigenvectors associated with each eigenvalue. The loadings can be matrix multiplied by either the original response matrix or the matrix of fitted values to locate each sample in two different ordination spaces:

- Site scores – result of multiplying loadings by the original variables. Stored as `Oak1_dbrda_summary$sites`.
- Site constraints – result of multiplying loadings by the fitted values (constrained axes) and residuals (unconstrained axes). Stored as `Oak1_dbrda_summary$constraints`. These are the data that are plotted when we graph a dbRDA.

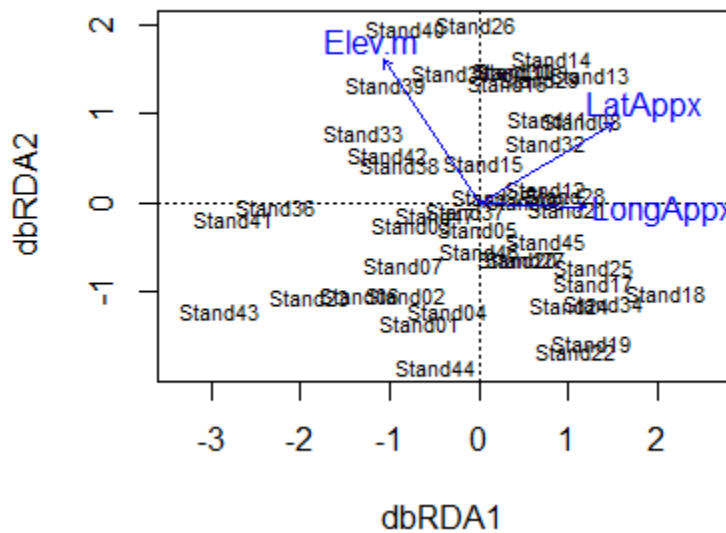
The constrained and unconstrained axes are uncorrelated with one another, as we saw previously with PCA and PCoA:

```
cor(Oak1_dbrda_summary$constraints) %>%
  round(2)
```

	dbRDA1	dbRDA2	dbRDA3	MDS1	MDS2	MDS3
dbRDA1	1	0	0	0	0	0
dbRDA2	0	1	0	0	0	0
dbRDA3	0	0	1	0	0	0
MDS1	0	0	0	1	0	0
MDS2	0	0	0	0	1	0
MDS3	0	0	0	0	0	1

To view the resulting ordination:

```
plot(Oak1_dbrda)
```



Notes:

- The first two axes are shown by default. If two or more axes are constrained, these are what will be shown. If the dbRDA was based on a single explanatory variable with one df (i.e., only one constrained axis), the constrained axis and the first unconstrained axis will be shown.
- The vectors show the direction in which each explanatory variable is most strongly associated with the axes.
- The location of each sample unit is shown by its name.

Unconstrained Axes

The unconstrained axes are the result of a PCA on the residuals (i.e., variation that was not related to the set of explanatory variables).

The unconstrained axes are named differently to distinguish them from the constrained axes. Furthermore, recall that a PCoA can produce negative eigenvalues if the underlying distance matrix does not satisfy the triangle inequality. As a result, two types of unconstrained axes are reported:

- MDS1, MDS2, etc. – unconstrained axes associated with positive eigenvalues
- iMDS1, iMDS2, etc. – unconstrained axes associated with negative eigenvalues (the 'i' refers to 'imaginary')

The unconstrained axes are generally not the topic of interest in a dbRDA. Do you see why?

Interpreting RDA and dbRDA

RDA uses Euclidean distances, so it is most appropriate when this distance measure is appropriate for the response matrix. If no constraining variables are included, the result is a regular PCA.

Although RDA and dbRDA begin with a multivariate linear regression linear model, Borcard et al. (2018, ch. 6.3.2.10) note that it is possible to also test second-degree (i.e., quadratic) variables. This can be useful if responses are unimodal, for example.

A major difficulty with all constrained ordinations is choosing which explanatory variables to constrain the ordination with. Model-testing and model-building functions can be applied to identify the optimal set of explanatory variables to include. See Borcard et al. (2018, ch. 6.3.2.6) for details. As more constraining variables are included, the model ends up increasingly behaving like an unconstrained ordination because there is more flexibility within the constraints.

Finally, note that the set of explanatory variables is considered *en masse*; the loadings demonstrate how the score on each constrained axis is a linear combination of those explanatory variables. These techniques are therefore not designed to evaluate which explanatory variables would be important if considered independently.

Conclusions

Several of these techniques are closely related. For example, a RDA without constraining explanatory variables is a PCA, and a dbRDA without constraining explanatory variables is a PCoA.

RDA does not have CCA's issues related to the use of the chi-square distance measure, though it does assume Euclidean distances and therefore is appropriate for variables where that distance measure is appropriate. dbRDA can be applied to a distance matrix obtained using any distance measure.

References

- Anderson, M.J., and T.J. Willis. 2003. Canonical analysis of principal coordinates: a useful method of constrained ordination for ecology. *Ecology* 84:511-525.
- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Legendre, P., and M.J. Anderson. 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs* 69:1-24.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.
- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McCordle, B.H. and M.J. Anderson. 2001. Fitting multivariate models to community data: a comment on distance-based redundancy analysis. *Ecology* 82:290-297.

Media Attributions

- dbRDA.biplot

40. CA, DCA, and CCA

Learning Objectives

To appreciate how CA and variations thereof work, and why these techniques can be contentious.

Exercises

Manly & Navarro Alberto (2017; ch. 12.5)

Examples

```
require(vegan, ggvegan)
```

Contents:

- Introduction
- The Chi-Square Distance Measure
- Correspondence Analysis (CA)
- Detrended Correspondence Analysis (DCA)
- Canonical Correspondence Analysis (CCA)
- Conclusions
- References

Introduction

Correspondence Analysis (CA), aka reciprocal averaging (Hill 1973), is an alternative to PCA for the analysis of sample × species data. Both techniques are eigenanalysis-based approaches. In addition, they both rely on a single matrix, and therefore are unconstrained or indirect gradient analysis methods. However, they differ in approach: PCA maximizes the variance explained by each axis, while CA maximizes the correspondence between species and sample scores.

Detrended Correspondence Analysis (DCA) and Canonical Correspondence Analysis (CCA) are extensions of CA. DCA, like CA, does not use a second matrix of explanatory variables and therefore is an indirect gradient analysis method. In contrast, CCA requires a second matrix of explanatory variables and therefore is a constrained or direct gradient analysis method.

These are very common analytical approaches, so it's important to be familiar with them. However, as you'll see throughout these notes, there are problems with CA and its extensions when applied to community data. That said, there is also disagreement in the literature about the degree to which these problems are 'fatal'. In my experience, for example, European ecologists tend to be more inclined to use these methods through software such as CANOCO (e.g., Šmilauer & Lepš 2014).

The Chi-Square Distance Measure

Theory

CA and its variants (DCA, CCA) are based on the same theory that contingency tests are based on. You may recall that a contingency table contains categorical rows and columns. Pearson's chi-square statistic is calculated by comparing the observed and expected values for each element in the table, where the expected values are calculated as:

$$\frac{y_{i\bullet} \times y_{\bullet j}}{y_{\bullet\bullet}}$$

where $y_{i\bullet}$ is the sum for row i , $y_{\bullet j}$ is the sum for column j , and $y_{\bullet\bullet}$ is the grand total.

CA preserves the chi-squared distance between sample units:

$$D_{i,j} = \sqrt{\sum_{k=1}^p \frac{y_{\bullet\bullet}}{y_{\bullet k}} \left[\frac{y_{ik}}{y_{i\bullet}} - \frac{y_{jk}}{y_{j\bullet}} \right]^2}$$

where

- $y_{\bullet\bullet}$ is the grand total
- $y_{\bullet k}$ is the total for species k (there are p species all together)
- $y_{i\bullet}$ and $y_{j\bullet}$ are the totals in sample units i and j
- y_{ik} and y_{jk} are the observed values for species k in sample units i and j

Note that the symbology in this equation is different from what was used in the chapter about distance measures.

(Aside: the chi-square distance can also be calculated using a formula that doesn't adjust for the grand total ($y_{\bullet\bullet}$); this is the formula in McCune & Grace (2002, p. 50). Legendre & Legendre (2012) consider these to be different distance measures: their D_{16} includes $y_{\bullet\bullet}$ while D_{15} does not.)

According to Wildi (2010), what is actually being compared in a chi-squared distance is not the raw data but the deviations from the expected values for each species in each sample unit.

The lower bound of the chi-square distance is zero for sample units that are identical. McCune &

Grace (2002), citing Minchin (1987), state that there is no upper bound for the chi-square distance, but Legendre & Legendre (2012) indicate that the upper bound is $\sqrt{2y_{..}}$.

We talked previously about standardizations, including the ‘standard’ adjustments by species maxima and site totals. Both of these standardizations are built into the Chi-square distance measure.

Chi-square distances are inherently standardized using **site totals** – note that the abundance of species k in the formula above is divided by the total abundance in each sample unit (i.e., $y_{i\bullet}$ and $y_{j\bullet}$). One implication of this is that all variables must be measured in the same scale because otherwise the site totals and grand total are rather nonsensical. If data aren’t measured in the same scale, all variables must be relativized (made dimensionless; on a column-by-column basis) before analysis.

Chi-square distances are also inherently standardized using **species totals** – note the $y_{\bullet k}$ in the denominator of the equation. This results in two issues with this distance measure:

- Changes in the abundance of a species on one sample unit affect the distances associated with that species on all sample units, even where it doesn’t occur.
- The weight given to each species is inversely proportional to its total abundance. In other words, high weight is given to species with low total abundance (i.e., rare species). If species are rare because they haven’t been adequately sampled, it may not be ecologically desirable to give them this much importance. Šmilauer & Lepš (2014) note that it is possible to downweight rare species. I haven’t explored this, but suspect one way to do so would be to standardize by species maxima before calculating chi-square distances.

Example

We will use a sample dataset we’ve used before (see ‘Simple Examples’ in the Distance Measures chapter) to explore chi-square distances:

Plot	SppA	SppB	SppC	SppD	SppE
1	1	1	1	0	0
2	0	0	0	1	1
3	1	1	1	1	1

(Source: Legendre & Legendre 2012, p. 311)

If we add the row and column totals to the matrix margins, we get the following:

Plot	SppA	SppB	SppC	SppD	SppE	$y_{i\bullet}$ or $y_{j\bullet}$
1	1	1	1	0	0	3
2	0	0	0	1	1	2
3	1	1	1	1	1	5
$y_{\bullet k}$	2	2	2	2	2	

By summing across either the row totals or column totals, you can verify that $y_{..} = 10$.

Use the chi-square distance formula (above) to calculate the chi-square distance between each pair of plots:

	Plot1	Plot2
Plot2		
Plot3		

Import the data (Legendre.Legendre.2012.p311.txt) into R and calculate the chi-square distance matrix. Functions such as `vegan::vegdist()` and `labdsv::dsvdis()` provide this capability.

```
test <- read.table("data/Legendre.Legendre.2012.p311.txt",
  header = TRUE)

library(vegan)

vegdist(test,
  method = "chisq") %>%
  round(3)
```

```
      Site1 Site2
Site2 2.041
Site3 0.816 1.225
```

Note that all of the species in this test dataset have the same species totals – the column totals are identical. You could create another test dataset to explore the contributions of common and rare species to the distance calculations.

We can easily use this dataset to explore how changes in the abundance of a species on one sample unit affect the distances among other sample units.

First, subset the data to just include plots 2 and 3 and then calculate the chi-square distance between the two plots. Compare this value to the value calculated above with the full dataset.

```
vegdist(test[2:3, ],
  method = "chisq") %>%
  round(3)
```

```
      Site2
Site3 1.212
```

Note that the distance between these two sample units has changed (1.225 to 1.212) even though the data haven't changed at all.

Second, change the abundance of one species on one plot and recalculate the entire distance matrix.

```
test %>%
  mutate(SppC = c(1, 0, 0)) %>%
```



```
vegdist(method = "chisq") %>%  
round(3)
```

```
      Site1 Site2  
Site2 2.062  
Site3 1.275 1.061
```

We altered a species in Site3, so it makes sense that the distances from that site to the other sites are affected. However, note that the distance from Site1 to Site2 is also affected (2.041 to 2.062) even though no abundances on either sample unit were changed!

Correspondence Analysis (CA)

Approach

Manly & Navarro Alberto (2017) note that a sample unit x species data matrix can be thought of as a two-way table of measures of abundance. This means that we can think about:

- The sample units in terms of the abundance of species they contain
- The species in terms of the sample units they occur in

Sample units that share the same species are more similar to one another than those that do not. By the same logic, species that occur in the same sample units are more similar to one another than those that do not. From this perspective, CA is a method for maximizing the correspondence between a sample unit-based classification and a species-based classification. The actual values that are calculated are inter-related: the values for the sample units are weighted averages of the species values while the values for the species are weighted averages of the sample unit values.

McCune & Grace (2002) describe how to conduct a CA using an eigenanalysis approach and a reciprocal averaging approach, and Manly & Navarro Alberto (2017) illustrate the connections between these two approaches. The eigenanalysis approach is the generally accepted technique. Its steps are:

1. Convert the sample unit x species matrix to a contingency table (**Q**) by calculating the contribution of each element to the Pearson chi-square statistic under the null model.
2. Obtain eigenvalues and eigenvectors from **Q**. This is done using the `svd()` function as described in the 'Matrix Algebra' notes. The formula being used is $\mathbf{X} = \mathbf{UDV}'$ where:

Matrix	Dimension	Notes
X	m x n	The matrix being analyzed.
U	m x n	Loadings (eigenvectors) for rows (samples) in original data matrix.
D	n x n	Diagonal matrix with singular values of X on diagonal. In other words, the eigenvalues. Eigenvalues are referred to as 'inertia' in CA and related techniques. As with PCA, the eigenvalues decline in magnitude. The first axis has the largest eigenvalue and maximizes the association between samples and species.
V	n x n	Loadings (eigenvectors) for columns (species) in original data matrix. Note: transposed in equation

Note that two eigenvectors (**U** and **V**) are produced for each eigenvalue – one related to the samples and the other to the species.

3. Use the loadings to calculate scores (coordinates) of sample units and species in ordination space. The second eigenvector is what allows CA to calculate coordinates for species.
4. Graph species and sample units on the same plot. This type of graph is known as a **bi-plot**, though it differs somewhat from the biplots we saw in our discussion of PCA.

CA in R: `vegan::cca()`

CA can be conducted using several different functions in R. In particular, the `vegan` and `ade4` packages both contain a function named `cca()`. Only one of these packages should be loaded at a time to avoid confusion. We'll use the version available in `vegan`. See the R help files for details about the usage of `cca()`.

Use the 'load.oak.data.R' script to load our oak data and make our standard adjustments to it.

```
source("scripts/load.oak.data.R")
```

To conduct a CA, we simply specify the name of the object containing the response variables:

```
ca.Oak1 <- cca(X = Oak1)
```

This function returns a large number of elements that are described in the help files (`?cca.object`) or by examining the object with `str()`. A few elements can be seen using the `summary()` function, and individual elements can also be referenced. Particular elements of interest:

- Eigenvalues: `eigenvals(ca.Oak1)` or `summary(eigenvals(ca.Oak1))[,1:6] %>% round(3)`

```

              CA1  CA2  CA3  CA4  CA5  CA6
Eigenvalue      0.514 0.335 0.279 0.251 0.237 0.226
Proportion Explained 0.102 0.066 0.055 0.049 0.047 0.045
Cumulative Proportion 0.102 0.168 0.223 0.272 0.319 0.364
```

- Site scores: `summary(ca.Oak1)$sites %>% round(3) %>% head()`

	CA1	CA2	CA3	CA4	CA5	CA6
Stand01	-0.400	0.797	-0.439	0.823	-0.382	0.740
Stand02	-0.330	0.435	0.256	0.278	-0.030	0.715
Stand03	0.040	0.561	-0.229	0.611	-1.337	0.749
Stand04	-0.066	0.441	-1.028	-0.491	0.003	-0.098
Stand05	1.756	-3.963	0.830	-1.110	0.002	0.928
Stand06	-0.501	0.121	-0.719	-0.722	-0.250	-1.386

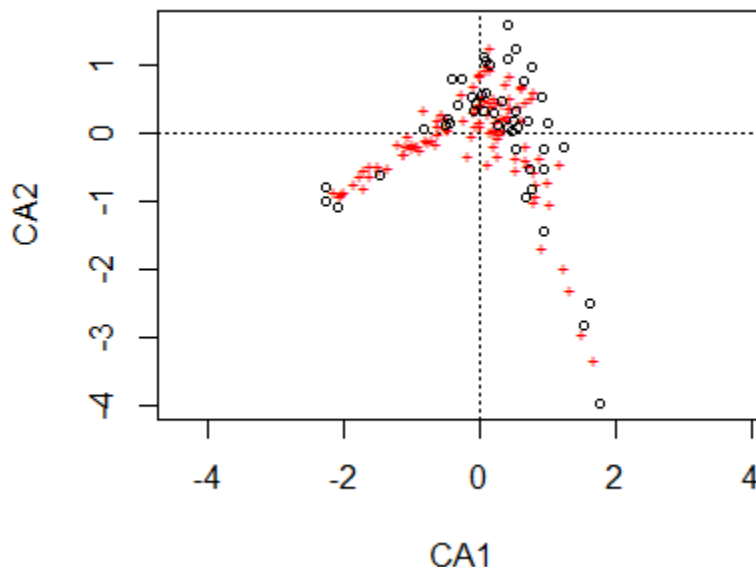
• Species scores: `summary(ca.Oak1)$species %>% round(3) %>% head()`

	CA1	CA2	CA3	CA4	CA5	CA6
Quga.t	0.214	0.058	-0.175	0.035	-0.037	0.061
Rhdi.s	0.119	0.316	0.081	0.103	-0.071	0.025
Syal.s	0.409	0.225	-0.597	-0.220	0.124	0.172
Amal.s	0.441	0.550	-1.087	-0.731	0.898	0.453
GAL	0.210	0.539	-0.036	0.351	-0.119	0.338
Povu	0.303	0.005	-0.128	-0.104	-0.084	-0.046

Note that the `summary()` function for this class of object only returns the first 6 scores for each species and site/sample unit. You can extract all of them from the original object.

To visualize a CA ordination in base R:

`plot(ca.Oak1)`



Note again that the `plot()` function is creating a particular type of graph based on the class of the object being plotted. The first and most obvious thing to note here is the horseshoe / arch effect, which we'll discuss below.

Sites are shown as circles in the graph; each is approximately located at the centroid of the points of the species that occur at that site. Species are shown as '+' symbols in the graph; each is at the optimum (peak of the assumed unimodal response curve) for that species.

Species and sites can be plotted as points or text using the `points()` and `text()` functions, respectively. They can also be plotted on separate graphs – I'll illustrate this using `ggplot2`.

Gavin Simpson has written functions to organize the objects created by `vegan` in a consistent manner for graphing via `ggplot2`. His functions are available in a package called `ggvegan`. – see the ['Visualizing and Interpreting Ordinations'](#) chapter for details on how to install it from github.

```
library(ggvegan)

autoplot(ca.Oak1)
```

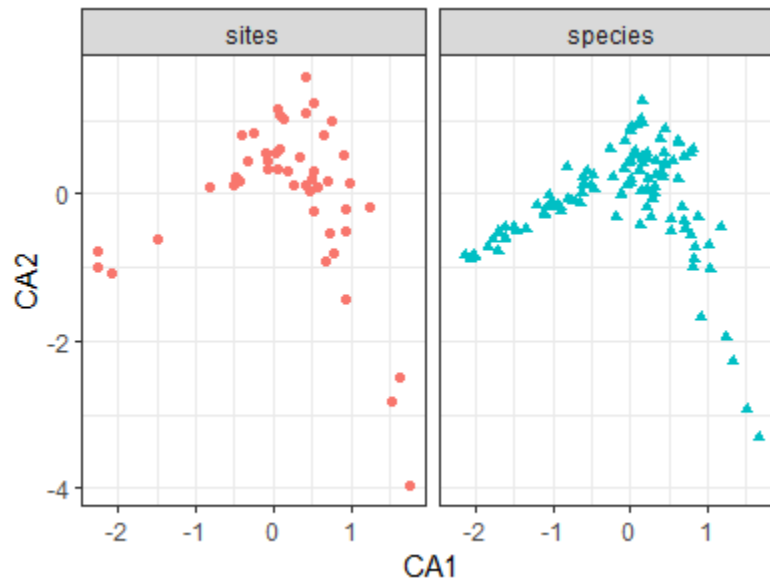


The `fortify()` function creates a dataframe that contains the necessary elements for graphing in `ggplot2`. This can be used to provide more customization than is available through `autoplot()`.

For example:

```
ca.Oak1.gg <- fortify(ca.Oak1)

ggplot(data = ca.Oak1.gg,
  aes(x = CA1, y = CA2)) +
  geom_point(aes(shape = Score, colour = Score)) +
  facet_grid(facets = . ~ Score) +
  guides(shape = FALSE, colour = FALSE) +
  theme_bw()
```



Interpretive Difficulties with CA

CA maximizes the separation of species abundances along each axis. Its primary advantages are that it:

- Yields a single outcome for a given dataset
- Permits species and sample units to be plotted as points on the same coordinate system.

However, McCune & Grace (2002) and Faith et al. (1987) give examples of how poorly CA is able to recreate synthetic gradients.

CA carries out a simultaneous ordination of the rows and columns of the data matrix. Thus, it can be difficult to draw inferences based on the association between the results of these ordinations.

CA assumes unimodal (i.e., bell-shaped) species response curves (ter Braak 1987). The optimum abundance of a species should be close to the point representing it in the ordination diagram, and its frequency of occurrence or abundance should decrease with distance from that point. However, this is not always the case:

- Species at the edge of the scatter plot may be absent from most sites – they show up near the sample(s) where they happened to be present.
- Species near the center of the ordination diagram may reach an optimum abundance in that area of ordination space. However, they may also have more than one optimum or be unrelated to the pair of ordination axes under consideration. It is not clear from the ordination which of these is the case. One way to assess this is to order the sample unit × species matrix with respect to sample and species scores on an ordination axis and then review this ordered matrix (ter Braak 1987).
- Species found away from the center of the diagram but not near the edges are most likely to display clear relationships with ordination axes.

Each axis is thought to represent a particular environmental gradient. However, 2nd and higher axes can be distorted if datasets span long gradients. This distortion classically shows up as an **arch (horseshoe)** in a CA (or PCA) ordination, and is clearly evident in the CA ordination of the Oakl

dataset. It is a mathematical artifact (Podani & Miklos 2002) and is thought to reflect the variability in species response curves.

Axis extremes can be compressed such that the spacing of samples along an axis doesn't reflect true differences in species composition.

Due to the above problems, McCune & Grace (2002) declare that “there should be no regular application of CA to ecological community data” (p. 154).

Detrended Correspondence Analysis (DCA)

Approach

DCA was developed by Hill & Gauch (1980) in response to several of the above problems with CA. It essentially is a CA followed by a few additional steps:

- **Detrending** to remove the arch effect – the first axis is divided into segments (default is 26; I'm not sure why) and the samples that fall within each segment are centered to have a mean of zero for the second axis.
- Nonlinear **rescaling** to correct compression of ends of gradients – segments are expanded or contracted so that the variation among species scores is equal among segments. As a result, species turnover is constant across the gradient. The units of the first DCA axis are now in species standard deviations (beta diversity), and the 'average' species is present along the gradient for about 4 standard deviations (McCune & Grace 2002, p. 30).

As a result of these changes, the eigenvalues are no longer directly interpretable (see 'Interpretive Difficulties with DCA' below). Instead, the effectiveness of a DCA ordination is calculated as the correlation between the distances among sample units in ordination space and in the original space.

DCA in R: `vegan::decorana()`

In R, DCA can be conducted using the `decorana()` function in the `vegan` package. See the help file for details. For example, arguments can be included to downweight rare species (give less weight to rare species; `iweigh`), specify the number of segments for detrending (default is 26; `mk`), etc.

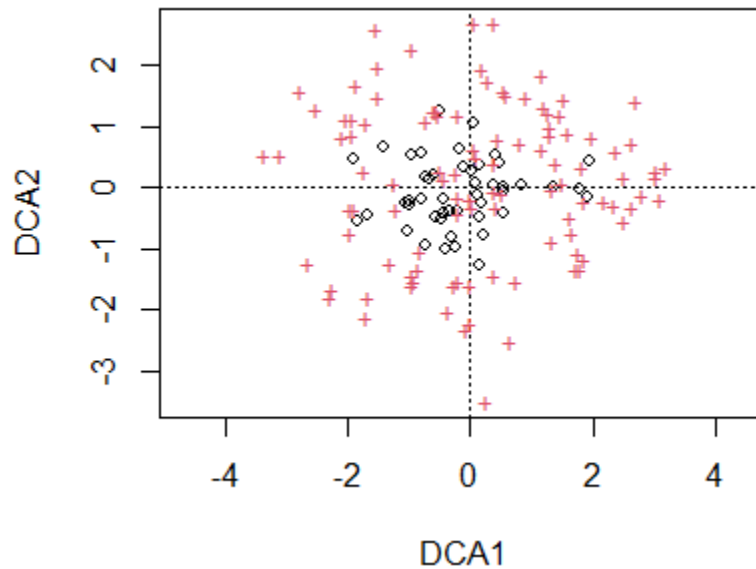
Applying this to our data:

```
dca.oak1 <- decorana(oak1)
```

Use `summary()` and related functions to see the results. Individual components can also be returned using indexing similar to that described for CA above. However, only the first four axes are reported.

To graph the results:

```
plot(dca.oak1)
```



Note that the arch present in the CA ordination is no longer present. Again, sites are indicated by circles and species by '+' symbols. We could use the code above to restrict our attention to sites or species, display points as text, etc.

Interpretive Difficulties with DCA

While DCA is able to remove the arch effect from an ordination, it does so in a rather arbitrary – or at least abstract – manner. After treatment in this fashion, it isn't entirely clear what the axes mean anymore. Also, detrending and rescaling have altered the eigenvalues such that they no longer represent the fraction of the variance represented by the ordination. As a result of these concerns, McCune & Grace (2002) and Gotelli & Ellison (2004) do not recommend the use of DCA.

Canonical Correspondence Analysis (CCA)

Approach

CCA (aka Constrained Correspondence Analysis) is a direct gradient analysis method. It was developed and popularized by ter Braak (1986, 1987). Like CA, it maximizes the correlation between species and sample scores. However, it does so **after** using multiple linear regression to constrain each sample unit to be a linear combination of the measured environmental variables. Because of this constraint, CCA yields lower eigenvalues than CA.

The process of conducting a regression followed by an ordination should remind you of Redundancy Analysis (RDA).

CCA in R: `vegan::cca()` Again

In R, CCA can be conducted using the same function as CA, except that an additional matrix is specified that contains the environmental data used to constrain the sample scores. See the help file for details.

We'll use a subset of the explanatory variables from the oak plant community dataset – in this case, the geographic variables (latitude, longitude, elevation). Since these are measured in different units, and have minimum values much greater than zero, we'll standardize them by their ranges before including them in the analysis.

```
geog.stand <- Oak %>%  
  select(LatAppx, LongAppx, Elev.m) %>%  
  decostand("range")
```

We can conduct the CCA in two ways. First, we can identify the response and explanatory objects separately:

```
cca(X = Oak1,  
    Y = geog.stand)
```

(note, confusingly, that the response matrix is argument `x` and the explanatory matrix is argument `y`).

Second, we can specify a formula relating the response to the explanatory variables:

```
cca.Oak1 <- cca(Oak1 ~ Elev.m + LatAppx + LongAppx,  
  data = geog.stand)
```

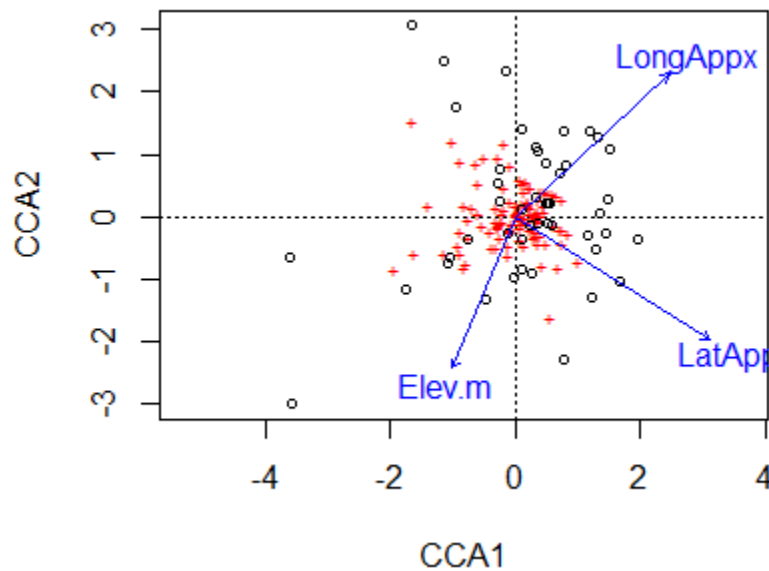
The formula specification is recommended as it provides the most flexibility.

As with CA and DCA, use `summary()` and related functions to see the results. Individual components of the results can be returned using the indexing described for CA above.

It is important to inspect the eigenvalues of the solution. Note that the constrained and unconstrained eigenvalues are distinguished from one another. There are as many constrained eigenvalues as there were explanatory variables in the constraining dataset; these are labeled as 'CCA1', 'CCA2', etc. and relate composition to those explanatory variables. The unconstrained eigenvalues ('CA1', 'CA2', etc.) are the eigenvalues calculated just as in a CA. Furthermore, note that the eigenvalues decline in value within the constrained and unconstrained sets, **but may jump up in magnitude as you move from the last constrained axis to the first unconstrained axis**. We saw this same behavior with RDA and dbRDA.

To visualize the first two constrained axes of this ordination:

```
plot(cca.Oak1)
```

Species or site scores can be displayed as described above for CA.

The resulting ordination can be tested for statistical significance using the `anova()` or `anova.cca()` function. The help file notes that this is actually a permutation test rather than an ANOVA, though the type of permutation test is not named. One of the first descriptions of this approach was provided by Legendre et al. (2011). Several options are available:

- `by = NULL` – test set of explanatory variables *en masse*, as was done in `adonis2()`.
- `by = "term"` – test terms in sequential order (sequential or Type I sums of squares), as was done in `adonis2()`.
- `by = "margin"` – test marginal effects (partial or Type III sums of squares), as was done in `adonis2()`.
- `by = "axis"` – test each constrained axis against the set of explanatory variables.

For example:

```
anova(cca.Oak1,
      by = "axis")
```

```
Permutation test for cca under reduced model
Forward tests for axes
Permutation: free
Number of permutations: 999

Model: cca(formula = Oak1 ~ Elev.m + LatAppx + LongAppx, data = geog.stand)
      Df ChiSquare      F Pr(>F)
CCA1    1    0.1842  1.7066  0.055 .
CCA2    1    0.1280  1.1859  0.416
CCA3    1    0.1076  0.9965  0.483
Residual 43    4.6420
```

```
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results indicate that the geographic variables are marginally related to the positions of sample units along the first constrained axis.

The `anova()` function can also be used to test the significance of other constrained ordinations, such as RDA.

There are also functions to test a suite of explanatory variables in a stepwise fashion. This can be done in a forward selection process via `add1.cca()`, in a backward selection process via `drop1.cca()`, or in an automatic process that does both forward and backward selection via `ordistep()` or `ordiR2step()`. Be advised that some people find model selection – whether in a multivariate or univariate context – to be helpful (e.g., Mitchell et al. 2017) but others dislike it.

Interpretive Difficulties with CCA

The first difficulty with CCA is how to select the environmental variables to include. How do you know that the important underlying environmental variables have been measured? Problems can arise from including too many or too few variables.

- Too many variables will result in relationships that appear to be strong yet are increasingly unconstrained. In fact, as the number of variables and number of sample units become similar, the results of CCA and CA become similar (and subject to the same methodological problems such as the arch effect).
- Too few variables may result in an analysis in which the sample units are not adequately represented as linear combinations of the variables.

Community structure that is unrelated to the environmental variables is ignored during the calculation of the constrained eigenvalues. This is why the first unconstrained eigenvalue may jump in magnitude – it is accounting for some of this structure (see `summary(eigenvals(cca.oak1))`). Note that the ratio of the eigenvalue for a particular axis to the total variation represented by the environmental variables is not a recommended statistic (McCune & Grace 2002, p. 168).

ter Braak (1987) noted that CCA is limited in the types of environmental data that can be included:

- **No ordinal variables** (e.g., moisture as low, moderate, or high levels). These types of data must be treated as if they were either continuous or nominal.
- **No missing values.** If an element of the environmental data is missing, either the entire sample unit must be deleted or the values must be estimated in some fashion.

However, I have not verified that this is still the case. There is no mention of these limitations in the R help files.

McCune & Grace (2002) suggest that CCA might be appropriate if:

- You are interested only in the aspects of community structure that relate to your measured

- environmental variables, and
- A unimodal model of species responses to these environmental variables is reasonable, and
- Chi-square distances are appropriate for your data.

Similarly, the `vegan` help file states that “CCA is a good choice if the user has clear and strong *a priori* hypotheses on constraints [i.e., about the explanatory variables] and is not interested in the major structure in the data set [i.e., in the response matrix]”.

Conclusions

CA and its variants are very popular in community ecology: von Wehrden et al. (2009) found DCA to be the most commonly used technique in their survey of several major journals. However, these techniques are dependent on assumptions that are unrealistic for most community-level data. The chi-square distance measure, for example, does not represent or recover synthetic gradients well (Faith et al 1987).

McCune & Grace (2002) note that “all uses of CA in the literature of community ecology that we have seen are either inappropriate or inadequately justified, or both” (p. 158).

The mathematical ‘solutions’ implemented in DCA are not very elegant. Once axes have been detrended and rescaled, they do not have a clear meaning. Borcard et al. (2018) mention the technique (Section 5.4.4) but, in light of its limitations, do not even illustrate it.

CCA focuses on known environmental gradient(s). It therefore requires prior knowledge about the gradient(s) and careful selection of which variables to use to constrain the ordinations. In light of the fact that chi-square distances are particularly sensitive to or affected by rare species, Borcard et al. (2018) suggest that “its use should be limited to situations where rare species are well sampled and are seen as potential indicators of particular characteristics of an ecosystem; the alternative is to eliminate rare species from the data table before CCA” (p. 256).

As evidenced by our other notes, there are other techniques that can provide insight into community ecology data without the problematic assumptions of CA and CCA, or the inelegant analytical solutions of DCA.

References

- Borcard, D., F. Gillet, and P. Legendre. 2018. *Numerical ecology with R*. 2nd edition. Springer, New York, NY.
- Faith, D. P., P.R. Minchin, and L. Belbin. 1987. Compositional dissimilarity as a robust measure of ecological distance. *Vegetatio* 69:57-68.
- Hill, M.O. 1973. Reciprocal averaging: an eigenvector method of ordination. *Vegetatio* 61:237-249.
- Hill, M.O., and H.G. Gauch, Jr. 1980. Detrended correspondence analysis: an improved ordination technique. *Vegetatio* 42:47-58.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.
- Legendre, P., J. Oksanen, and C.J.F. ter Braak. 2011. Testing the significance of canonical axes in redundancy analysis. *Methods in Ecology and Evolution* 2:269–277.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.

- Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Minchin, P.R. 1987. An evaluation of the relative robustness of techniques for ecological ordination. *Vegetatio* 69:89-107.
- Mitchell, R.M., J.D. Bakker, J.B. Vincent, and G.M. Davies. 2017. Relative importance of abiotic, biotic, and disturbance drivers of plant community structure in the sagebrush steppe. *Ecological Applications* 27:756-768.
- Podani, J., and I. Miklos. 2002. Resemblance coefficients and the horseshoe effect in principal coordinates analysis. *Ecology* 83:3331-3343.
- Šmilauer, P., and J. Lepš. 2014. *Multivariate analysis of ecological data using CANOCO 5*. 2nd edition. Cambridge University Press, Cambridge, UK.
- ter Braak, C.J.F. 1986. Canonical correspondence analysis: a new eigenvector technique for multivariate direct gradient analysis. *Ecology* 67:1167-1179.
- ter Braak, C.J.F. 1987. The analysis of vegetation-environment relationships by canonical correspondence analysis. *Vegetatio* 69:69-77.
- von Wehrden, H., J. Hanspach, H. Bruelheide, and K. Wesche. 2009. Pluralism and diversity: trends in the use and application of ordination methods 1990-2007. *Journal of Vegetation Science* 20:695-705.
- Wildi, O. 2010. *Data analysis in vegetation ecology*. Wiley-Blackwell, West Sussex, UK.

Media Attributions

- CA
- CA.gg
- CA.gg2
- DCA
- CCA

41. Comparison of Ordination Techniques

Learning Objectives

To consider how to choose among ordination methods.

To demonstrate graphical and analytical methods for comparing ordination techniques, both informally and through Procrustes analysis.

Readings (Recommended)

von Wehrden et al. (2009)

Key Packages

```
require(vegan, tidyverse)
```

Introduction

Recall that ordinations are used for two primary purposes:

- Data reduction (i.e., reducing dimensionality)
- Visualization

Ordinations are not used for statistical tests – constrained ordinations blur the boundary between the two, though they don't provide significance tests. Instead, it is most helpful to think of all ordinations as visual support for the statistical tests that have been conducted. They can also help generate hypotheses that are then investigated in future studies.

We noted at the start of this section that the 'ideal' ordination does not exist. And, as you may

have noticed from the readings (e.g., von Wehrden et al. 2009) and from your conversations with others, people have their own preferences about which ordination technique is preferred. In my opinion, the appropriate ordination technique is the one whose assumptions your data most closely correspond to.

Here, I briefly review the assumptions and perceived pros and cons of the techniques that we've covered.

Eigenanalysis Techniques

PCA, CA, and related ordinations are based on eigenanalysis techniques:

- Based on the sample unit \times species data matrix
- One eigenvalue per axis, with axes explaining declining proportions of the variance
- Each eigenvalue has associated eigenvectors
- The score for each sample unit along an axis is the sum of the products of the loadings (elements in the eigenvector) and variables (species).

PCA

PCA has the most restrictive assumptions (Euclidean distance measure, multivariate normality, linear relationships among variables) but is also the foundational ordination technique. It is highly appropriate for some types of data, such as morphological/physiological data (e.g., Summerville et al. 2006) and LiDAR data (e.g., Kane et al. 2010), but is inappropriate for community-level data (Minchin 1987; Legendre & Legendre 2012).

Other situations where PCA is particularly useful include:

- Translating and rotating data, as in the adjustments to the final solution in some NMDS ordinations, including `metaMDS()`. Note that, in this scenario, we keep all axes and therefore do not reduce the dimensionality. As a result, the only effect of this procedure is to efficiently organize the data points in a configuration that can ease interpretation.
- Combining correlated variables to reduce the dimensionality of a dataset. One or more of the resulting principal components can then be used as covariates in subsequent analyses (e.g., Haugo et al. 2011).

CA, DCA, and CCA

CA and related techniques have assumptions that are difficult to justify for community-level data (chi-square distance measure, unimodal species response curves). They plot sample units and species on the same graph (a biplot), but the interpretation of these points may not be straightforward. For example, species on the edges of the scatter plot may be rare and those in the center may be unrelated to the axes, so the focus should be on species at intermediate distances from the centroid. In addition, there's the arch effect that shows up in CA (and can also occur in PCA and CCA). Some folks advocate for DCA (Šmilauer & Lepš 2014) while others strongly advocate against it (e.g., McCune & Grace 2002; Gotelli & Ellison 2004).

Distance-based Techniques

NMDS

NMDS “preserves the rank order of the inter-point dissimilarities as well as possible within the constraints of a small number of dimensions” (Anderson et al. 2008, p. 105). It has the least restrictive assumptions (any distance measure, no assumptions about shape of species response curves).

NMDS doesn't provide quite the same types of information as other techniques – particularly because the axes have no direct meaning. The user has to choose the number of dimensions, which can sound arbitrary or intimidating. However, analyses can be rerun with different dimensions to identify the effect of dimensionality on stress. And, the procedure is computationally intensive, though this concern is diminishing as computing power increases. Species can be located in the ordination space using a different method than CA and DCA, though the same criticisms of such species scores apply.

PCoA

PCoA has parallels to both PCA and NMDS. It is somewhat more restrictive than NMDS because it relates the distances in ordination space to the actual distances in the distance matrix (not the ranks of the distances). However, PCoA has the flexibility that it can be used with any distance measure while projecting the points onto axes like PCA. It is conceptually more similar to PERMANOVA than any of the other indirect gradient analysis techniques covered here.

PCoA is the default method for identifying the initial starting coordinates of a NMDS and for expressing sample unit locations in a Euclidean space in PERMDISP.

Constrained or Direct Gradient Analysis Techniques

Constrained techniques (CCA, RDA, dbRDA) are amalgamations or extensions of the other techniques. They assume that you understand – *a priori* – the controlling gradient in your ecosystem. Variability that is not related to the chosen environmental variables is ‘ignored’ (actually, it's deferred to the unconstrained eigenvalues).

CCA makes the same assumptions as CA and DCA about the appropriateness of the chi-square distance measure, etc. (though I see it frequently in the literature, perhaps because it is an older technique).

RDA is appropriate if Euclidean distances are reasonable for your data, while dbRDA is appropriate with any distance measure. Both of these techniques are amalgamations of other techniques.

One instance where RDA or dbRDA can be useful is as a follow-up to a statistical analysis in which you have identified an explanatory variable that relates to composition. For example, dbRDA “will display a cloud of multivariate points by reference to a specific *a priori* hypothesis” (Anderson & Willis 2003, p. 511). These techniques can be applied to data expressed with the same distance measure as was used in analysis, and will illustrate the differences due to the explanatory variables as strongly as possible. See the following section for an example of this.

Different Views of the Same Point Cloud: Comparing Ordinations

Ordinations are often used with complex multivariate data. The high dimensionality of these data implies that there are many ways to view them – imagine looking at a cloud of data points from different perspectives. Furthermore, because ordinations are not statistical tests, it is appropriate to explore your data in different ways. Let's illustrate this by visualizing the relationship between grazing status and community composition in two complementary ways:

- Using NMDS to see differences between grazing treatments within the context of the 'overall' variability in composition
- Using dbRDA to see the maximal distinction between the grazing treatments

We'll focus on the relationship between grazing history – the combination of current grazing status (GrazCurr) and past grazing status (GrazPast) – and the Oak1 response matrix.

Use the `load.oak.data.R` script to load and make initial adjustments to the oak plant community dataset. The resulting `Oak1` object contains abundances of the 103 most abundant species, each relativized by its maxima, while `Oak_explan` contains the potential explanatory variables.

```
source("scripts/load.oak.data.R")
```

Combine the two elements of grazing history together in a single column within `Oak_explan` (the object containing the explanatory variables). For later use, I will also create a column containing the stand ID.

```
Oak_explan <- Oak_explan %>%  
  unite(col = Grazing, c(GrazPast, GrazCurr), remove = FALSE) %>%  
  mutate(Stand = rownames(Oak_explan))
```

PERMANOVA

Test whether plant community composition differs with respect to grazing status:

```
adonis2(Oak1 ~ Grazing,  
  data = Oak_explan,  
  distance = "bray")
```

```
Permutation test for adonis under reduced model  
Terms added sequentially (first to last)  
Permutation: free  
Number of permutations: 999  
  
adonis2(formula = Oak1 ~ Grazing, data = Oak_explan, distance = "bray")  
      Df SumOfSqs      R2      F Pr(>F)  
Grazing  2    0.9095 0.07844 1.8725  0.007 **  
Residual 44   10.6854 0.92156  
Total    46   11.5949 1.00000  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```


Yes, Grazing has a significant effect on plant community composition.

Conduct Ordinations (NMDS, dbRDA)

Conduct a NMDS of these data –:

```
set.seed(42)

Oak1_nmds <- metaMDS(comm = Oak1,
  autotransform = FALSE,
  k = 3)
```

I specified a 3-dimensional solution but did not adjust other NMDS arguments such as `maxit`, `try`, etc. I also set the random number generator before running it to ensure that my output is repeatable. Recall that this is unconstrained and does not incorporate grazing information.

Now, conduct a dbRDA relating oak plant community composition to grazing status:

```
Oak1_dbrda <- dbrda(Oak1 ~ Grazing,
  data = Oak_explan,
  distance = "bray")

print(Oak1_dbrda)
```

```
Call: dbrda(formula = Oak1 ~ Grazing, data = Oak_explan, distance
= "bray")
```

	Inertia	Proportion	Rank	RealDims
Total	11.59490	1.00000		
Constrained	0.90946	0.07844	2	2
Unconstrained	10.68544	0.92156	44	33

Inertia is squared Bray distance

Eigenvalues for constrained axes:

dbRDA1	dbRDA2
0.7987	0.1108

Eigenvalues for unconstrained axes:

MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
1.5778	1.1950	1.0434	0.8402	0.7163	0.6529	0.5248	0.4470

(Showing 8 of 44 unconstrained eigenvalues)

Compare the ANOVA table from the PERMANOVA and the summary of the dbRDA. The `sumOfSqs` in the former is equivalent to the `Inertia` in the latter, and the proportion of variance associated with each term is identical in the two tables.

Organize Coordinates

Extract the coordinates from each ordination. To graph the ordinations together, we need to

combine their coordinates in a single object. However, the coordinates are indexed in different ways and named differently for each ordination. We will use a simple function to extract them, give them the same names (X, Y), and add columns distinguishing the type of ordination (Type) and identifying the stand (Stand).

Create function:

```
extract.coordinates <- function(xx, type) {  
  data.frame(X = xx[,1],  
    Y = xx[,2],  
    Stand = rownames(xx),  
    Type = type)  
}
```

Note that `xx` is the set of coordinates from the object produced by the ordination, not the object itself.

Apply this function to each ordination object:

```
Oak1_nmds_gg <- extract.coordinates(  
  xx = Oak1_nmds$points,  
  type = "NMDS")  
  
Oak1_dbrda_gg <- extract.coordinates(  
  xx = summary(Oak1_dbrda)$sites,  
  type = "dbRDA")
```

Combine the two objects together:

```
NMDS_v_dbrDA <- rbind(Oak1_nmds_gg,  
  Oak1_dbrda_gg)
```

There are 94 rows in this object because each stand is represented twice – once from the NMDS and once from the dbRDA.

Add the grazing status to this object:

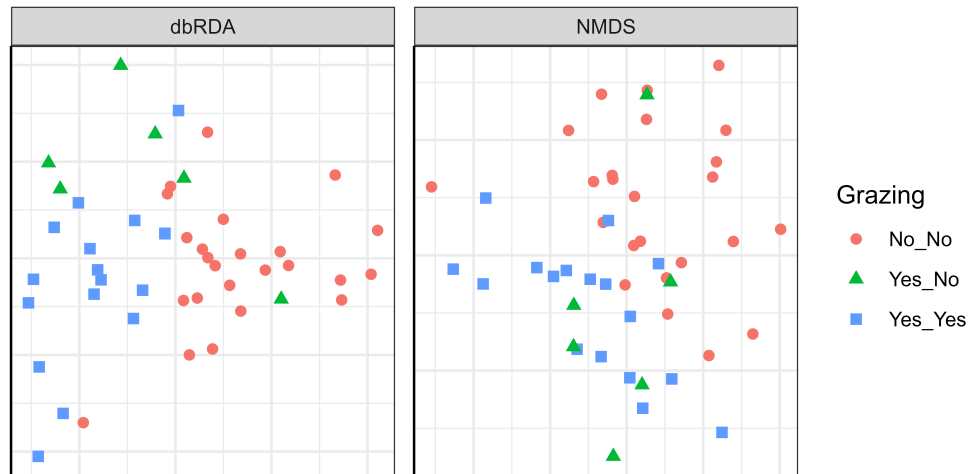
```
NMDS_v_dbrDA <- merge(x = NMDS_v_dbrDA,  
  by = "Stand",  
  y = Oak_explan[, c("Stand", "Grazing")])
```

Create Graphs

Plot the ordinations beside each other:

```
ggplot(data = NMDS_v_dbrDA, aes(x = X, y = Y)) +  
  geom_point(aes(shape = Grazing, color = Grazing), size = 2) +  
  labs(shape = "Grazing", color = "Grazing") +  
  facet_wrap(vars(Type), scales = "free") +  
  theme_bw() +  
  theme(axis.line = element_line(),
```

```
axis.ticks = element_blank(),
axis.text = element_blank(),
axis.title = element_blank())
```



Two ordinations of the same data. The dbRDA ordination shows as much of the grazing-related composition variation as can be represented in two dimensions. The NMDS ordination shows as much of the total compositional variation as possible in two dimensions.

These two ordinations look different yet are based on exactly the same data! Some notes:

- The grazing treatments are not in the same positions within each ordination, but this does not hinder interpretation.
- We did not include axis labels here. It would be appropriate to do so for the dbRDA (including the amount of variation accounted for by each axis), but we generally would not do so for NMDS ordinations.
- In the NMDS ordination, sample units are located so that the data cloud accounts for as much of the variation in composition as possible.
 - The NMDS solution was rotated via PCA so that the axes account for diminishing amounts of the variation.
 - Grazing treatment was not incorporated into the ordination. We superimposed it onto the ordination *a posteriori* by coding each sample unit by its grazing treatment.
 - Although grazing had a statistically significant effect on composition, it only accounted for 7.8% of the total variation in composition (see PERMANOVA results). It therefore is not surprising that there is considerable overlap between the grazing treatments.
- In the dbRDA, we looked at the data cloud from the perspective that showed the maximal distinction between the grazing treatments.
 - There are three grazing treatments, so both visible axes are constrained. The eigenvalues (`summary(Oak1_dbrda)$cont`) show that the first and second constrained axes account for 6.89% and 0.96%, respectively, of the variation in composition. Together, these axes represent all possible variation that can be explained by grazing (i.e., $6.89 + 0.96 = 7.85\%$).
 - Note the 'danger' of misinterpreting the ordination ... the axes have equal lengths here but we have to remember that the second axis accounts for a small amount of the total variation. How much? $0.96 / 7.85 = 12.2\%$ of the variation that can be explained by grazing status. Therefore, the first axis is accounting for 87.7% of the variation that can be explained by grazing status.

Finally, please note that it is appropriate to consider, and even to display, different ordinations of the same data. As evidenced here, these ordinations can give complementary insights into the structure of the data.

Procrustes Analysis

The above graphical comparison is visually striking, but what if we want to compare two ordinations statistically? One way to do so is called **Procrustes analysis** (Peres-Neto & Jackson 2001). (Aside: the story of Procrustes is a memorable one!)

Briefly, in Procrustes analysis we hold one ordination constant and then superimpose the other ordination onto it. The superimposed ordination is rotated and scaled so that it corresponds as closely as possible to the other ordination. The Procrustes statistic is the sum of squared differences between the positions of each sample unit in the two ordinations. A helpful feature of Procrustes analysis is that it also permits the user to assess the alignment between individual observations, not just for the dataset as a whole (Peres-Neto & Jackson 2001).

Procrustes analysis can be conducted using the `vegan::procrustes()` function, or it can be conducted and the significance (non-randomness) of the relationship between the ordinations simultaneously assessed using the `vegan::protest()` function.

Let's apply this to the two ordinations that we conducted above. The two key arguments are the two ordinations to be compared; See the help file for other arguments in this function.

```
procrustes.eg <- protest(Oak1_nmds, Oak1_dbrda)
```

```
print(procrustes.eg)
```

```
Call:
protest(X = Oak1_nmds, Y = Oak1_dbrda)

Procrustes Sum of Squares (m12 squared):      0.6749
Correlation in a symmetric Procrustes rotation: 0.5702
Significance: 0.001

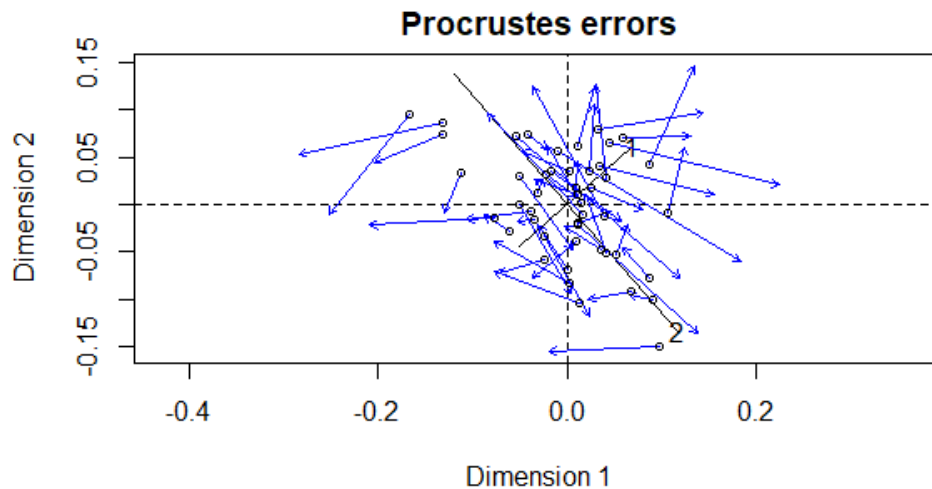
Permutation: free
Number of permutations: 999
```

Note that we accepted the default of 999 unrestricted permutations. More complex designs could be incorporated by restricting permutations using the `how()` function that we discussed in the context of group comparison tests.

The results indicate that there is a significant alignment between the two ordinations, although the correlation is somewhat low (0.57).

More helpful than the statistic itself is a visualization of the patterns:

```
plot(procrustes.eg)
```



Procrustes analysis of the oak compositional data, relating NMDS and dbRDA ordinations of the stands.

Notes:

- There are two sets of axes, one related to each ordination.
- The first set of axes (dashed lines) are from the first ordination that we specified – `oak1_nmds`.
- The second set of axes (solid lines, with '1' and '2' labeling the two axes) are from the second ordination that we specified – `oak1_dbrda`. Note that axis 1 from this ordination is actually the smaller of the two axes when overlaid onto the NMDS ordination – this is an indication of how much the data cloud was rotated to align with the Grazing term.
- The symbols are the coordinates of the sample units in the rotated or second ordination (in this case, the dbRDA).
- Each sample unit has an associated blue arrow that points from its location in the second ordination to its location in the first ordination (the NMDS). This highlights another benefit of this image – showing us which points in different ordinations represent the same sample unit.

Post-script: Negative Eigenvalues

The above comparison could also be made with RRPP as the statistical test. However, `lm.rrpp()` automatically adjusts for negative eigenvalues – see the notes about this technique for details. **If you are going to visualize data that have been adjusted in this fashion during analysis, the ordination should also be adjusted in the same way.** If you don't do this, the analysis and the visualization are not fully based on the same data.

Adjustments for negative eigenvalues are easily done in a dbRDA; see the 'add' argument in the `dbrda()` function.

The `metaMDS()` function does not include an adjustment for negative eigenvalues. One way to do so would be to conduct a PCoA with this adjustment, save all axes, and then conduct a NMDS on the PCoA dimensions.

Conclusions

There are a wide range of ordination techniques available, though they can be organized into broad themes. In addition, some techniques are closely related (e.g., PCA is a special case of PCoA), and some techniques are amalgamations of others (e.g., dbRDA is a combination of PCoA, RDA, and PCA).

In my opinion, NMDS and dbRDA are some of the most robust techniques that we've covered. A particularly appealing feature of these techniques is that they can be used to visualize a dataset using the distance measure as were used in a statistical analysis. Furthermore, dbRDA follows the same logic as PERMANOVA.

The relative differences among some of these techniques are unclear; Anderson (2015) notes that techniques like NMDS and PCoA will tend to give similar results, and that far more important differences arise from decisions about data adjustments (transformations, standardizations) and which distance measure to use.

I've stressed that ordinations primarily are visualization tools. They therefore are often most appropriately thought of in a supplementary role – for example, supporting a statistical analysis by demonstrating a statistically significant pattern that has been detected.

In some cases, ordinations can be very effective means of data reduction. This is particularly true for PCA, where the principal components can be related to the original variables and can be analyzed independently even though the original variables were correlated with one another.

A few wise quotes to conclude:

"A biologist's insight, experience, and knowledge of the literature are the most important tools for interpreting indirect gradient analysis" (source missing).

"... when the dust settles, if you've found a pattern in NMDS, you will probably find something similar with DCA/CCA and maybe even with PCA. When you see someone's presentation which contains a DCA ordination or CCA, there's probably good information in it ... and shouting, "Your ordination sucks, man!" probably won't get you deep into an informative conversation." (Mike Kearsley, 2003, unpublished).

References

- Anderson, M.J., R.N. Gorley, and K.R. Clarke. 2008. *PERMANOVA+ for PRIMER: guide to software and statistical methods*. PRIMER-E Ltd, Plymouth, UK.
- Anderson, M.J., and T.J. Willis. 2003. Canonical analysis of principal coordinates: a useful method of constrained ordination for ecology. *Ecology* 84:511-525.
- Gotelli, N.J., and A.M. Ellison. 2004. *A primer of ecological statistics*. Sinauer, Sunderland, MA.
- Haugo, R.D., C.B. Halpern, and J.D. Bakker. 2011. Landscape context and long-term tree influences shape the dynamics of forest-meadow ecotones in mountain ecosystems. *Ecosphere* 2:art91.
- Kane, V.R., R.J. McGaughey, J.D. Bakker, R.F. Gersonde, J.A. Lutz, and J.F. Franklin. 2010. Comparisons between field- and LiDAR-based measures of stand structural complexity. *Canadian Journal of Forest Research* 40:761-773.
- Legendre, P., and L. Legendre. 2012. *Numerical ecology*. Third English edition. Elsevier, Amsterdam, The Netherlands.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Minchin, P.R. 1987. An evaluation of the relative robustness of techniques for ecological ordination. *Vegetatio* 69:89-107.

Peres-Neto, P.R., and D.A. Jackson. 2001. How well do multivariate data sets match? The advantages of a Procrustean superimposition approach over the Mantel test. *Oecologia* 129: 169-178.

Šmilauer, P., and J. Lepš. 2014. *Multivariate analysis of ecological data using CANOCO 5*. 2nd edition. Cambridge University Press, Cambridge, UK.

Summerville, K.S., C.J. Conoan, and R.M. Steichen. 2006. Species traits as predictors of lepidopteran composition in restored and remnant tallgrass prairies. *Ecological Applications* 16:891-900.

von Wehrden, H., J. Hanspach, H. Bruelheide, and K. Wesche. 2009. Pluralism and diversity: trends in the use and application of ordination methods 1990-2007. *Journal of Vegetation Science* 20:695-705.

Media Attributions

- NMDS.v.dbRDA
- Procrustes.nmds.dbrda

42. General Graphing Principles

Learning Objectives

To introduce the `quick.metaMDS()` function which encodes desired default settings for `metaMDS()` arguments.

To review how to create graphics with base R capabilities, including symbol shapes, colors, and combining multiple graphics in a single figure.

To review how to create graphics with `ggplot2`, including adjusting aesthetics, themes, and using faceting to create separate but related graphics.

To identify ways to save graphics as digital files.

Key Packages

```
require(vegan, tidyverse)
```

Introduction

One of the advantages of software like R is that it can perform sophisticated analyses yet also has extremely powerful graphical capabilities. This allows us to conduct our statistical analyses and to customize the ordinations that visualize patterns.

These notes include examples using the base R graphing capabilities and the (more powerful) features of `ggplot2`. As you will glimpse here, virtually every element of a graph can be customized.

Aside: The `quick.metaMDS()` Function

We'll use a NMDS ordination to illustrate the general graphing principles. We saw in the NMDS chapter that the `metaMDS()` function includes many arguments. We may want to change arguments from their defaults (e.g., `autotransform`), but we also often want to keep many of these arguments the same within our code. Rather than typing them out every time, we can more efficiently hard code them in a function:


```
quick.metaMDS <- function(dataframe, dimensions, n.try = 40) {
  require(vegan)
  metaMDS(comm = dataframe,
    autotransform = FALSE,
    distance = "bray",
    engine = "monoMDS",
    k = dimensions,
    weakties = TRUE,
    model = "global",
    maxit = 300,
    try = n.try,
    trymax = 100,
    wascores = TRUE)
}
```

The arguments that change in this function are shown in red font. To use this function, we need to specify `dataframe` and the desired number of `dimensions`. The `n.try` argument refers to the minimum number of starts from new random coordinates. This argument includes a default of 40 – we can overwrite this value if desired (e.g., `n.try = 50`) but if we don't specify this argument the function will execute with 40 starts from new random coordinates.

To adjust other aspects of the NMDS ordination, we can either:

- Permanently change them within the code of the function (e.g., edit function code so that `trymax = 200`, or so that `distance = "euclidean"`)
- Include them in the set of arguments associated with `quick.metaMDS()`

The `quick.metaMDS()` function is available in the GitHub repository. Save it to the 'functions' subfolder within the folder where your R project is located.

Oak Example

We'll use the oak plant community dataset to illustrate graphical procedures. Use the `load.oak.data.R` script to load and make initial adjustments to the oak plant community dataset. The resulting object, `oak1`, contains abundances of the 103 most abundant species, each relativized by its maxima.

```
source("scripts/load.oak.data.R")
```

We'll conduct a NMDS ordination of the `oak1` object and use that as our example below. We'll load the `quick.metaMDS()` function and then call it:

```
source("functions/quick.metaMDS.R")
```

```
Oak1.z <- quick.metaMDS(dataframe = Oak1,
  dimensions = 3)
```

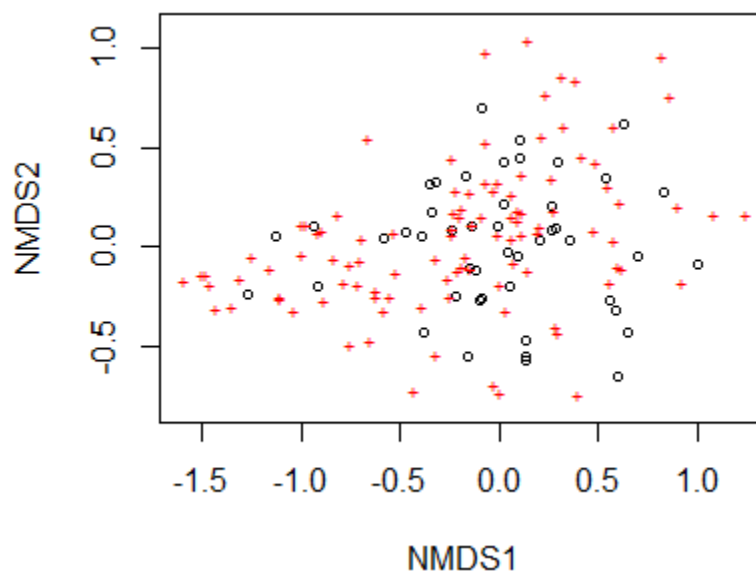
Base R Graphing

The base R graphing capabilities are helpful to know in their own right, but are also helpful because they are the basis for other approaches such as `ggplot2`. For example, the shapes assigned to points in `ggplot()` are identified using the numerical codes shown below for plotting character (`pch`).

For more information about general plotting and graphics in R, see Venables & Ripley (2000), Dalgaard (2008), Murrell (2006), and Sarkar (2008). In addition, there are numerous helpful websites (e.g., <https://www.statmethods.net/graphs/index.html>).

As we've seen numerous times, R contains functions whose actions depend on the class of the object being manipulated. For example, the `plot()` function can be applied to class 'metaMDS' objects:

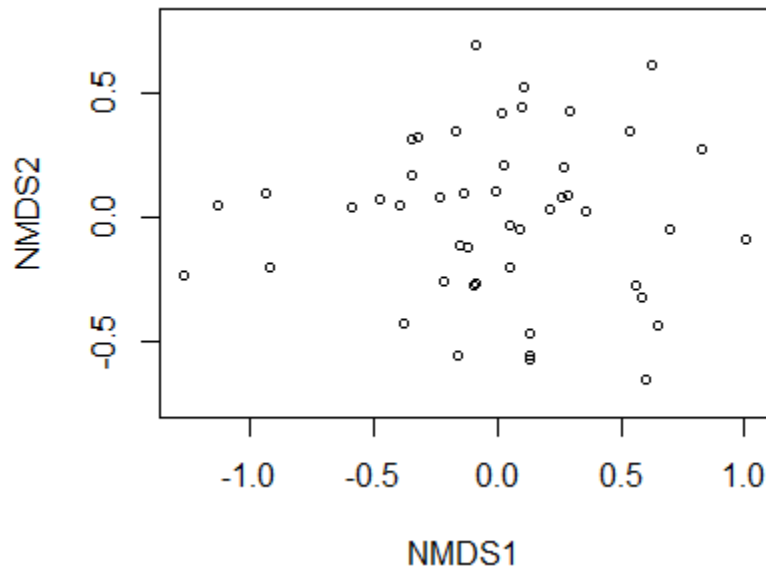
```
plot(Oak1.z)
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by open circles and species centroids by red '+' signs.

We accepted all defaults during this action; one of those defaults is to plot both the site coordinates and species coordinates. We can specify that we only want to display the site coordinates:

```
plot(Oak1.z,  
     display = "sites")
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by open circles.

Plotting Functions


To customize graphics, it is helpful to break the graphing down into a series of functions. Several commonly used functions are shown in the following table. See the help files for arguments and details.

Function	Purpose
<code>plot()</code>	Plot graph. Can specify whether to plot with points (default; <code>type = "p"</code>), textual labels (<code>type = "t"</code>), or axes only (no data; <code>type = "n"</code>). The latter type gives you control over the points and other elements via subsequent functions.
<code>points()</code>	Plot points onto existing graph.
<code>text()</code>	Add text to existing graph.
<code>lines()</code>	Add line(s) to existing graph.
<code>legend()</code>	Add legend to existing graph.
<code>title()</code>	Add title(s) to existing graph.
<code>abline()</code>	Add reference lines to existing graph. Can specify the intercept (a) and slope (b) of the line, the y-value for a horizontal line (h), and the x-value for a vertical line (v).

Key Plotting Arguments

Each of the above functions has associated arguments as described in the help files. However, they also have a `'...'` at the end of the list of arguments. This means that we can also specify additional arguments. In particular, we can draw on the arguments from other graphing functions. The

following – and many more – arguments are available; see the help files for functions such as `par()`, which sets global parameters, for details.

Argument	Description
main	Text to print as main title for the figure.
sub	Text to print as a sub-title for the figure.
cex	Scaling for symbols and text. Default is 1; 1.5 is 50% larger while 0.5 is 50% smaller.
pch	<p>Plotting character – either an integer specifying a symbol or a single character to be used as the default in plotting points. The ‘examples’ section of the help file for the <code>points()</code> function includes code to produce this visual list of options:</p> <p style="text-align: center;">plot symbols : points (... pch = *, cex = 2.5)</p> 
col	Default plotting color. See below for details.
xaxt	Axis type for horizontal axis: "n" = none; "s" = standard (default)
yaxt	Axis type for vertical axis: "n" = none; "s" = standard (default)
xlim	The limits of the horizontal axis. Unless otherwise specified, R automatically sets the limits of x to be slightly larger than the minimum and maximum values of the data being plotted. Specify as <code>xlim = c(x1, x2)</code> . Note that <code>x1 > x2</code> is allowed and leads to a ‘reversed axis’.
ylim	The limits of the vertical axis. Unless otherwise specified, R automatically sets the limits of y so that they are slightly larger than the minimum and maximum values of the data being plotted. Specify as <code>ylim = c(y1, y2)</code> . Note that <code>y1 > y2</code> is allowed and leads to a ‘reversed axis’.
xlab	Text to print as the label on the horizontal axis.
ylab	Text to print as the label on the vertical axis.
lty	<p>Line type:</p> <ul style="list-style-type: none"> • 0 = blank • 1 = solid (default) • 2 = dashed • 3 = dotted • 4 = dotdash • 5 = longdash • 6 = twodash
lwd	Line width: default is 1, 2 is twice as thick.
asp	Aspect ratio, calculated as y/x.
pin	A vector giving the current plot dimensions, (<code>width,height</code>), in inches.

Font style:

- font
- 1 = plain
 - 2 = bold
 - 3 = italic
 - 4 = bold italic
 - 5 = symbol

family Font family. The actual font may differ depending on whether you are running R on Windows or Mac. In windows, the default options are "serif" (Times New Roman), "sans" (Arial), "mono" (Courier New), and "symbol" (Symbol).

ps Font point size. Default is 16. Text size = `ps * cex`.

Choosing Colors

Colors are an extremely important aspect of visualizations. They can be identified in multiple ways:

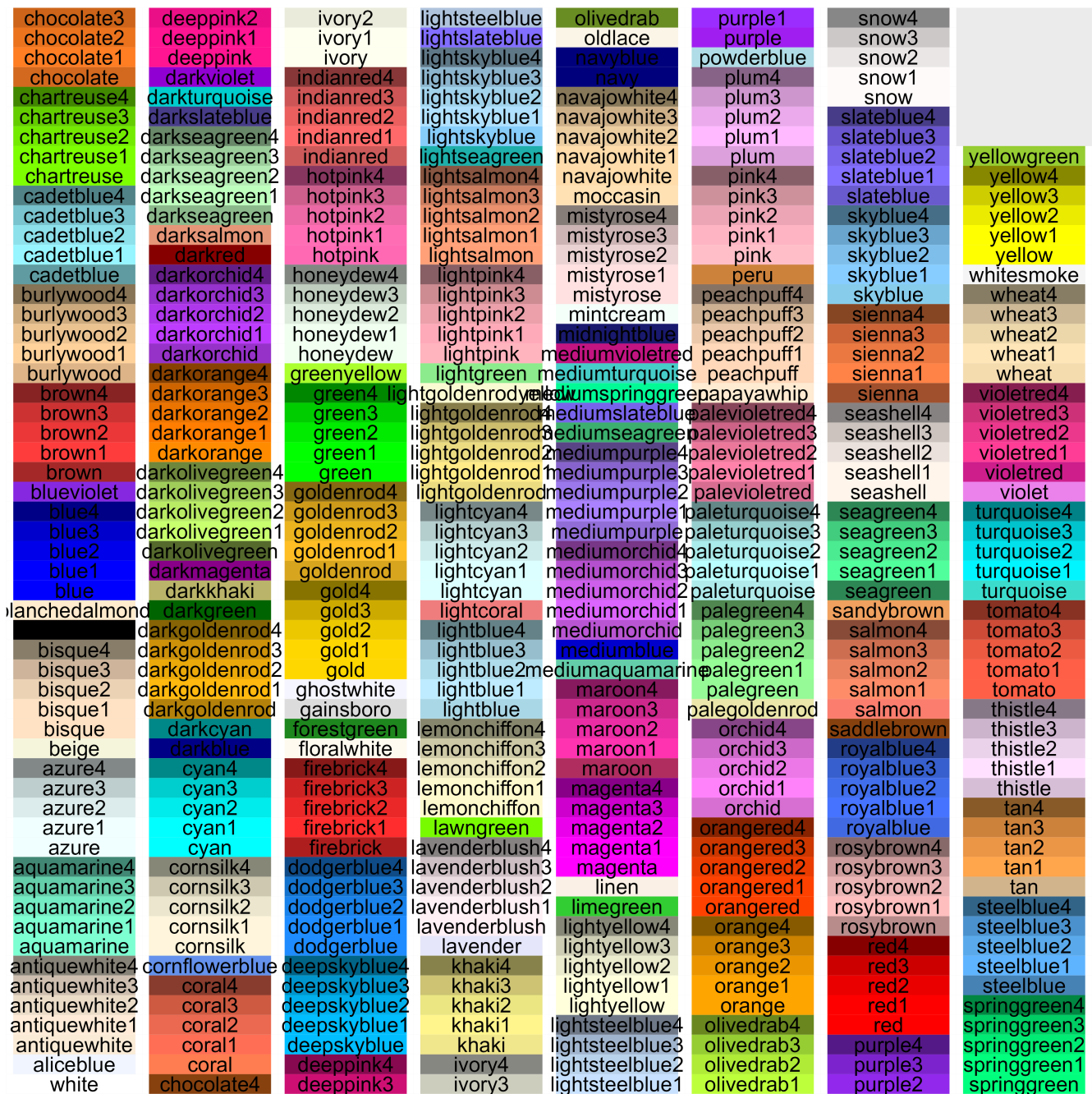
- By integer number, for each named color
- By name, for each named color (see below)
- By hexadecimal string for many more colors. Each hexadecimal string has the form "#RRGGBB" where each of the pairs RR, GG, BB consists of two hexadecimal digits giving a value in the range 00 to FF for that color (red, green, blue).

For example, 'chartreuse' can be specified as `colors()[47]` or `col = "chartreuse"` or `col = "#7FFF00"`.

In recent versions of R, the name or code of a specified color is highlighted in that color when it is typed in a script or in the Console.

Named Colors

There are 657 named colors within R. 224 of these are shades of grey and gray (either spelling is accepted). This image identifies the non-grey/gray colors by name:



I created the above image using the following code, which is tweaked from that available at <http://sape.inf.usi.ch/quick-reference/ggplot2/colour>. That website also includes nice graphics showing other aspects of color (hue, saturation, chroma, RGB combinations, etc.).

```
color.list <- data.frame(color = colors()) %>%
  filter(! str_detect(color, "grey|gray"))

d <- data.frame(color = color.list,
  y = seq(0, nrow(color.list)-1) %% 55,
  x = seq(0, nrow(color.list)-1) %% 55)
```

```
ggplot(data = d) +
  scale_x_continuous(name="", breaks=NULL, expand=c(0, 0)) +
  scale_y_continuous(name="", breaks=NULL, expand=c(0, 0)) +
  scale_fill_identity() +
  geom_rect(aes(xmin=x, xmax=x+1, ymin=y, ymax=y+1), fill="white") +
  geom_rect(aes(xmin=x+0.05, xmax=x+0.95, ymin=y, ymax=y+1, fill=color)) +
  geom_text(aes(x=x+0.5, y=y+0.5, label=color),
    colour="black", hjust=0.5, vjust=0.5, size=3)

ggsave("graphics/colors.png", width = 6.5, height = 6.5,
  units = "in", dpi = 600)
```

Color Palettes (ColorBrewer)

It is very common to need multiple colors to distinguish groups in a graphic. The ColorBrewer website (<http://colorbrewer2.org/>) is a valuable source of information about this.

ColorBrewer provides color palettes for a large number of scenarios, including sequential or diverging or qualitative classes. You simply specify how many classes you want to show. In addition, you can filter palettes based on other criteria such as:

- Colorblind safe
- Print friendly – suitable for color printing
- Photocopy safe – suitable for greyscale printing or photocopying

The colors within the selected palette are identified on the screen by their hexadecimal string.

If using `ggplot2`, palettes can be called by name using `scale_color_brewer()` and `scale_fill_brewer()`.

An Example in Base R

We can use these arguments to build a custom figure. We will do so in stages – first we'll create an empty plot with no axes – this is important because doing so scales the axes according to the coordinates of the sites even though the units are not displayed on either axis. Once we've done that, we'll add a title and individual points using separate functions:

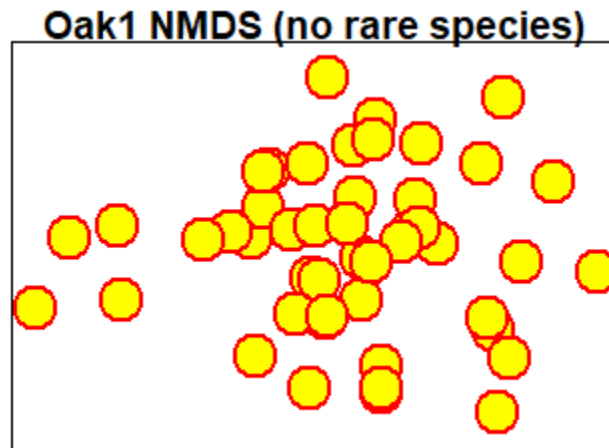
```
plot(Oak1.z,
  display = "sites",
  type = "n",
  xlab = "",
  xaxt = "n",
  ylab = "",
  yaxt = "n")

title(main = "Oak1 NMDS (no rare species)")

points(Oak1.z$points,
  pch = 21,
  cex = 3,
  col = "red",
```



```
bg = "yellow",
lwd = 2)
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by yellow circles.

Textual Labels

When textual labels are added to an ordination, the graph can easily become difficult to read where points are in close proximity and labels therefore overlap. Several *vegan* functions help with this problem:

- `ordilabel()` – uses opaque labels; can choose a variable with which to prioritize the order in which labels are assigned.
- `ordipointlabel()` – automatically optimizes the location of the labels to improve readability:
`ordipointlabel(Oak1.z, display = "sites")`
- `orditorp()` – adds text only where it does not cover already-present labels, and points otherwise.
- `orditkplot()` – produces an ordination in which you can manually select and move labels to improve readability. You can then save the output as an EPS file or dump it into R, where it is saved as an object that you can then plot.
`orditkplot(Oak1.z, display = "sites")`
 (note that this function opens a new graphical window)

Combining Multiple Graphs in a Single Figure

In addition to customizing an individual graph, we can build single figures that contains multiple graphs (or other images).

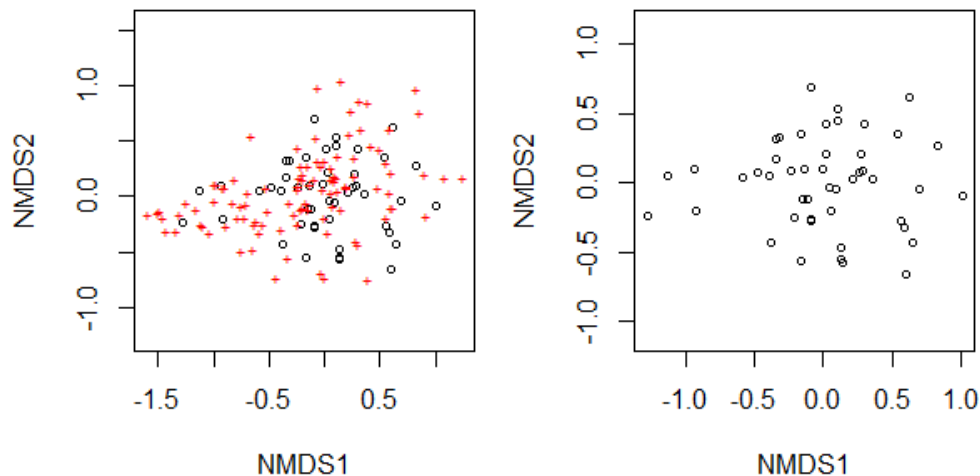
The `mfc01` and `mfcrow` arguments within `par()` are particularly important in this regard. Both arguments require a vector in the form `c(nr, nc)` specifying the number of rows (`nr`) and columns (`nc`) to create; each element in this matrix of figures will be a separate image. The arguments differ in terms of whether graphs are drawn working down the columns (`mfc01`) or across the rows (`mfcrow`).

To use this capability, you execute `par()` first and specify either `mfcol` or `mfrow`. You can then draw the figures themselves.

```
par(mfcol = c(1,2))
```

```
plot(Oak1.z)
```

```
plot(Oak1.z, display = "sites")
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by open symbols in both graphics. The red plus signs in the left graphic indicate the positions of species centroids within the ordination space. The scales vary slightly between the two graphics.

These changes to the graphical window remain in effect until the graphic window is closed or `mfrow/mfcol` is reset:

```
par(mfcol = c(1,1))
```



Graphing with `ggplot2`

The `ggplot2` package is, in my opinion, one of the game-changers within R. It has changed the way many people think about and create graphics. Once you are familiar with its structure, it is much more powerful and versatile than the base R plotting capabilities. I highly recommend it.

Chang (2013) focuses on this package, and the help files for this package are online (<https://ggplot2.tidyverse.org/reference/>) and include many visual examples. Another source of information is chapter 1 from Wickham & Golemund (2017). Finally, the package's cheatsheet

(<https://rstudio.github.io/cheatsheets/data-visualization.pdf>) provides a compact visual summary of its capabilities.

As a measure of its broad appeal, dozens of other packages have been written that organize data for use in `ggplot2` or that automate the production of particular types of graphics within the `ggplot2` 'universe'. A few of these are highlighted in the chapter about visualizing and interpreting ordinations.

The Grammar of Graphics

In `ggplot2`, graphical objects have a particular 'grammar'. Each graphic can be described as a series of components. Figures 1 and 2 below (from Wickham 2010) illustrate how a graphical object can be created by combining these elements.

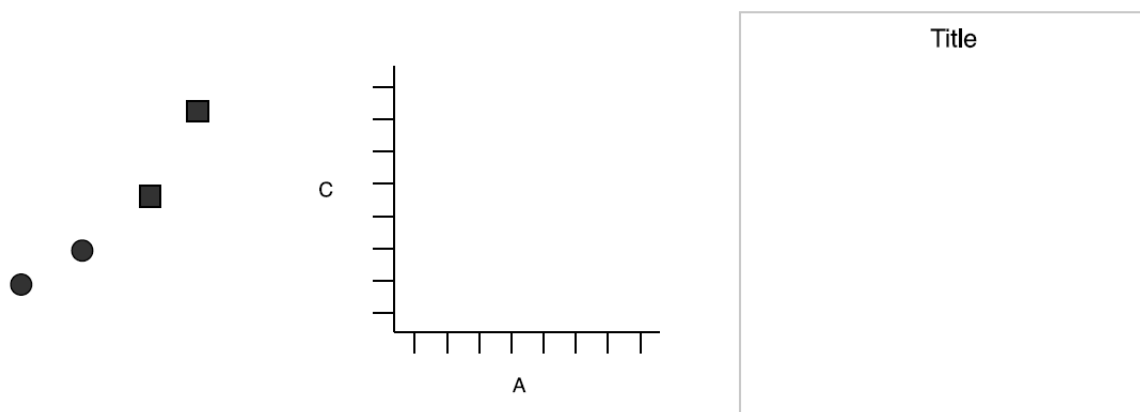


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

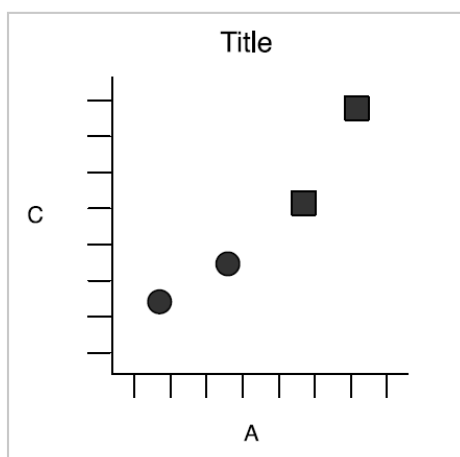


Figure 2. The final graphic, produced by combining the pieces in Figure 1.

The advantage of this grammar is that each element can be controlled separately. For example, the geometric object can be changed from points to lines without having to change other elements of the graphic. Similarly, a bar chart and a pie chart differ simply in their coordinate system.

The functions that provide other capabilities in `ggplot2` are in several classes, including:

- `geoms` – geometric objects, such as whether to plot the data as points, lines, bars, etc.
- `guides` – axes and legends
- `scales` – how to customize appearance of geoms
- `facets` – multiple panels in same graph
- `themes` – formatting of axes, panel background, etc.

For example, the `ggplot2` cheatsheet provides this template:

```
ggplot(data = <Data>) +  
  <Geom_Function>(mapping = aes(<Mappings>),  
    stat = <Stat>,  
    position = <Position>) +  
  <Coordinate_Function> +  
  <Facet_Function> +  
  <Scale_Function> +  
  <Theme_Function>
```

The elements in bold font are required; all others are optional.

We will create a simple graphic and then use it as a template while covering other aspects of `ggplot2` grammar.

`ggplot2::ggplot()`

The workhorse function is `ggplot()` – this is where the data are identified and the broad ‘structure’ of the graph (e.g., variables that will define the x and y axes) is established. The elements are then modified or added in other functions. Functions are linked together by adding a trailing ‘+’ after each function – this way, the whole set of functions is executed as a unit. This approach also makes it easy to selectively turn elements off by commenting the relevant line out.

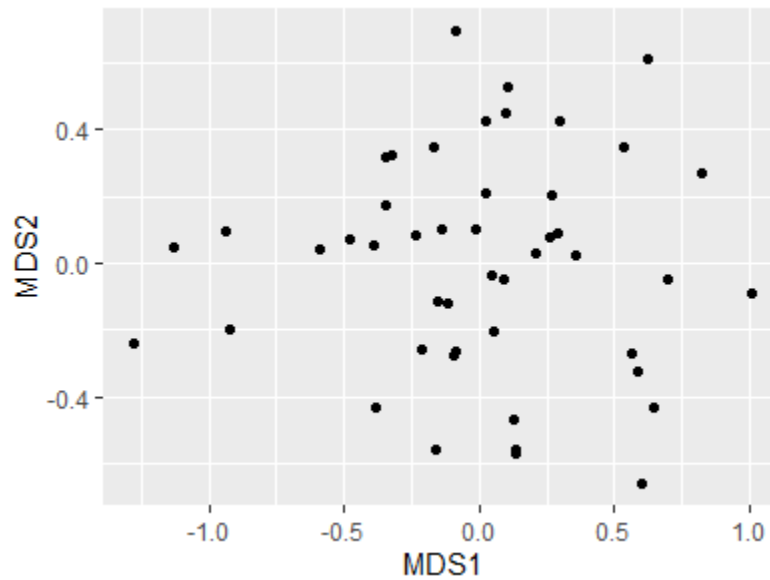
Here is an example, focusing just on the site scores. For convenience, we begin by combining these scores with the `Oak_explan` dataframe:

```
Oak_explan <- merge(x = Oak_explan,  
  y = Oak1.z$points,  
  by = "row.names")
```

```
library(ggplot2)
```

```
p <- ggplot(data = Oak_explan,  
  aes(x = MDS1, y = MDS2)) +  
  geom_point()
```

```
p
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by black points.

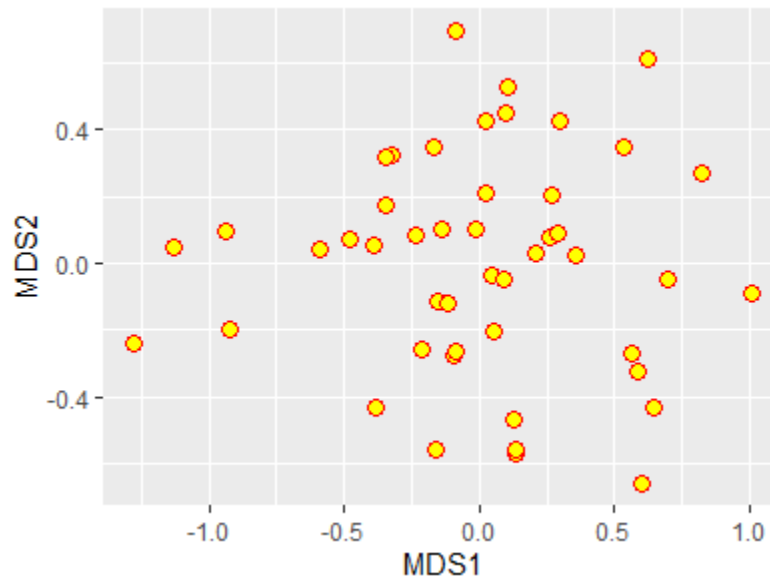
Note that we saved the above graph to an object (p); this allows us to call p and tweak it without having to repeat those lines of code.

Aesthetics

Every element of a graphic can be customized. Elements can be 'hard-coded' (applied consistently throughout) or can vary with the data. This distinction can be confusing when learning `ggplot2`: elements that vary with the data must be referenced within the `aes()` (aesthetic) function within that element.

For example, we can redraw p with the same color scheme as in the base plotting above:

```
p +
  geom_point(size = 3,
             shape = 21,
             color = "red",
             fill = "yellow")
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by yellow points.

In this case, I changed the symbology in the image printed on the screen but I did not change the object `p`.

However, if we wanted the symbol color to vary amongst the levels of a grouping variable, we would specify the color within the `aes()` function and assign it to that grouping variable:

```
p + geom_point(aes(color = GrazCurr))
```

Scales

The colors that are assigned to the levels can be specified by including a `scale_color_*` function. The `*` in this function name reflects the fact that there are many options, including:

- `scale_color_brewer()` – use one of the pre-defined ColorBrewer color schemes.
- `scale_color_discrete()` – assign a different color to each level, using a default color scheme.
- `scale_color_manual()` – manually specify the colors to be used. Assigned to the levels in the order that they are recognized in the data.

For example if we want to manually set the colors for the two levels of our current grazing status variable:

```
p + geom_point(aes(color = GrazCurr)) +  
  scale_color_manual(values = c("red", "blue"))
```

This color scheme is shown in the graphic that is saved to a file near the end of this chapter.

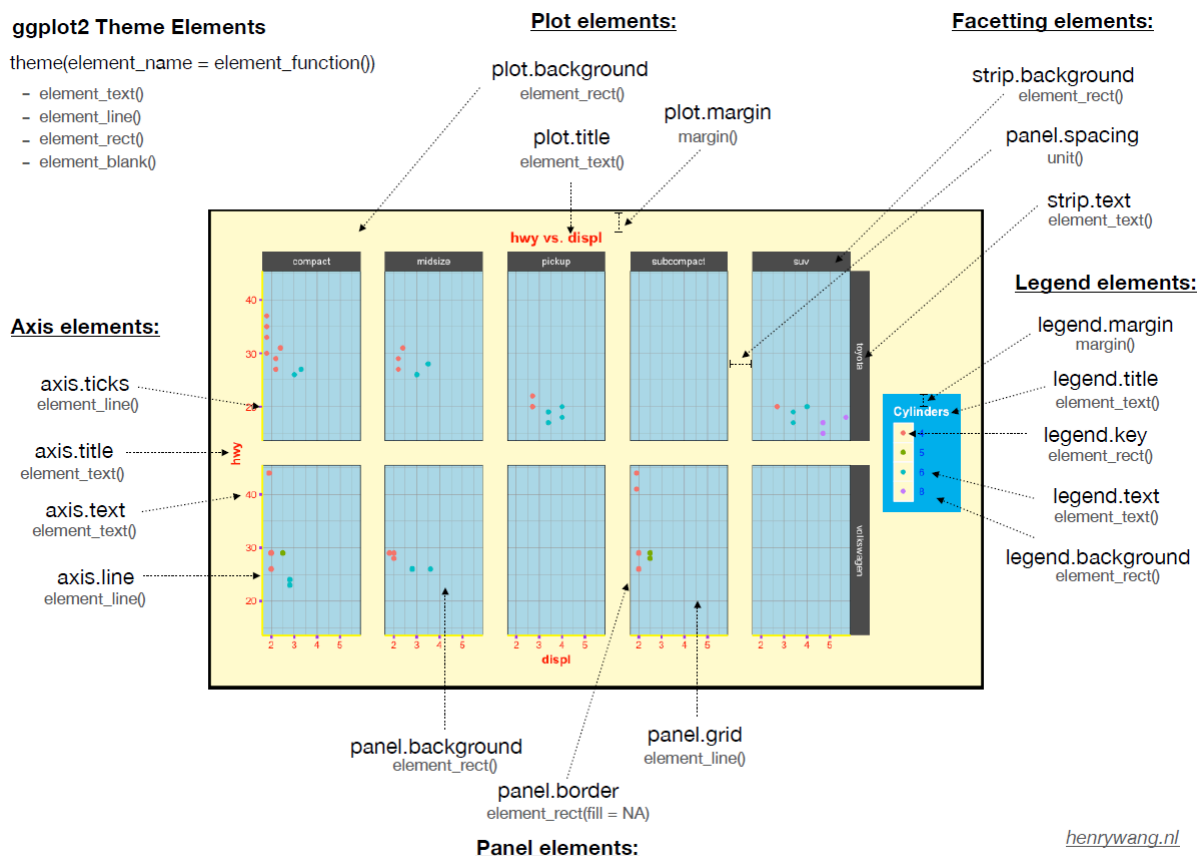
Similar options exist for other aesthetic parameters, including:

- `scale_fill_*`()
- `scale_linetype_*`()
- `scale_linewidth_*`()
- `scale_shape_*`()
- `scale_size_*`()

See the help files for more information about these and other `scale_*`() functions.

Themes

Every element of a `ggplot2` graphic can be controlled. The visual appearance of the graphic is set through the theme, which controls its many elements:



henrywang.nl

Derived from "ggplot2: Elegant Graphics for Data Analysis"

Theme elements in `ggplot2`. Image from <https://henrywang.nl/ggplot2-theme-elements-demonstration/>.

Personally, I like graphs that have a white background and dark borders. There is a pre-defined theme, `theme_bw()`, for this in `ggplot2`. Pre-defined themes can be used directly or combined with additional customizations. For example, we can omit the axis labels and numerals as is common when drawing a NMDS ordination.

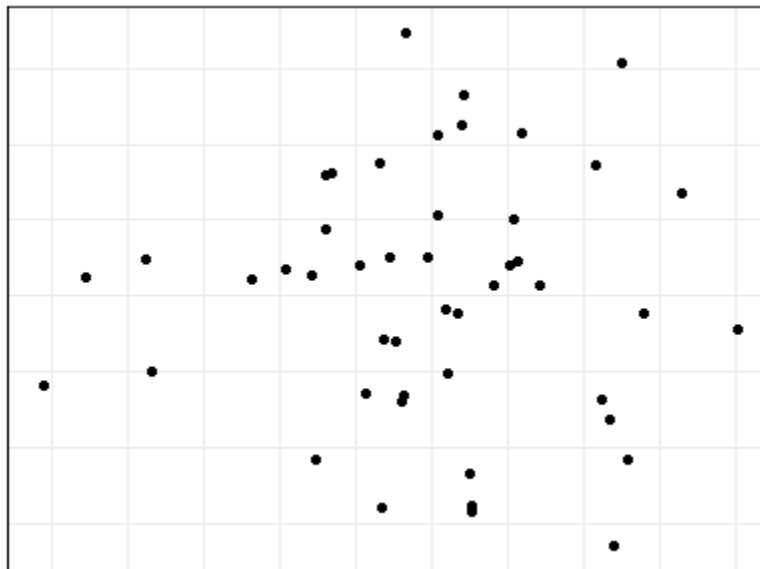
Furthermore, typing out all aspects of the code every time is cumbersome and can easily lead to inconsistencies among graphics. One way to create consistent graphics is to create an object that contains our desired settings for `theme()`:

```
theme.custom <- theme_bw() +  
  theme(axis.line = element_line(),  
        axis.ticks = element_blank(),  
        axis.text = element_blank(),  
        axis.title = element_blank())
```

Note that `theme.custom` is an object, not a function, and therefore should not have parentheses after it.

Now, we can call this object whenever we want to apply these thematic settings to a graphic:

```
p +  
  theme.custom
```



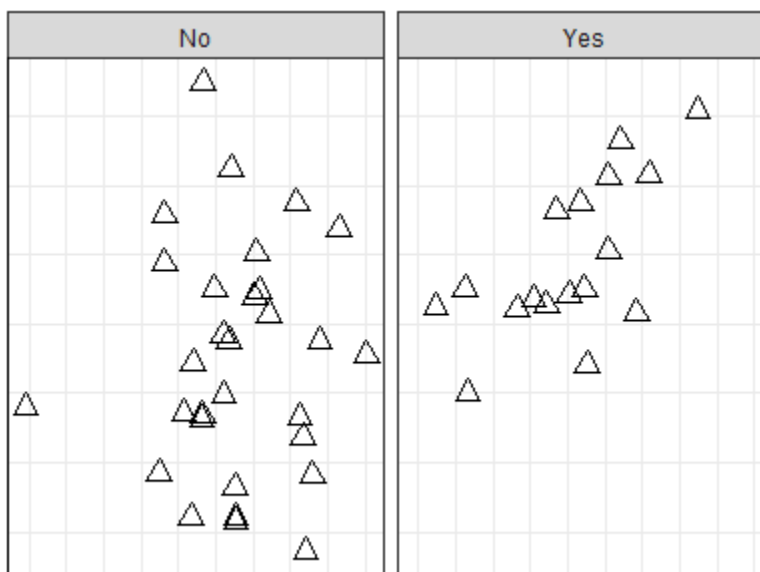
First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by black points.

Another feature of `ggplot2` is that there are intelligent default values, and those defaults are hierarchical. For example, we suppressed both axes in `theme.custom` above. To suppress just the `x` axis, the arguments within `theme()` can be changed to `'*.x'`. Try adjusting `theme.custom` by changing the `axis.text` argument to `axis.text.x` and re-creating the above image – it will now show the vertical axis but not the horizontal axis.

Combining Multiple Graphs in a Single Figure

In `ggplot2`, multiple graphs are easily combined using **faceting**. Faceting simply creates a separate graph for each level of the grouping variable. For example, suppose we want to view the current grazing statuses separately:

```
ggplot(data = Oak_explan, aes(x = MDS1, y = MDS2)) +  
  geom_point(size = 3, shape = 2) +  
  facet_grid(cols = vars(GrazCurr)) +  
  theme.custom
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by open triangles. Although all stands were part of a single ordination, they are graphed separately based on whether or not they experienced grazing when data were collected.

These graphics are based on the same data object and have the same scales for each axis.

We can easily create very different graphics just by choosing whether and how to facet. Faceting can be done by columns (`cols`; as was done above), or by rows, or both. A single variable with a large number of levels can be shown via the `facet_wrap()` function.

There are also arguments that allow scales to vary among facets.

The `geofacet` package allows you to facet in a manner that mimics the original geographic parameters. For example, state-level data could be faceted with the facets approximating the actual arrangement of the states on the earth's surface.

Other packages build on the `ggplot2` platform to provide additional features for creating more complex sets of graphics. Examples include the `cowplot` and `ggarrange` packages.

Saving Graphics

There are several ways to save graphics produced in R.

First, you can export images directly from the RStudio Plots window. This opens a pop-up window in which you can choose the file type, image name, and image size (width and height, in pixels). This can work well for a small number of graphs but is laborious if you are creating multiple graphs.

Second, you can zoom in on the graph from the RStudio Plots window, manually resize it if desired, and then right click and choose either 'Save image as' or 'Copy Image'. This is more convenient, but it is hard to ensure that multiple graphs are the same size.

Third, you can directly save the image to a file that you specify, while also hard-coding the dimensions that you want. The file will be saved in the working directory as specified in your R project. The code to do so differs between base R and `ggplot`.

Saving Graphics in Base R

Different functions exist to save the image in different graphical formats: `bmp()`, `jpeg()`, `png()`, and `tiff()`. The basic approach is to:

- Create a 'graphic device' with the file name and formatting instructions
- Plot the image
- Close the graphic device via `dev.off()`

Since the image is plotted directly to the file and not displayed, you have to open and inspect it. You may have to tweak settings and re-run your script to get the graphic to display as desired.

For example, let's save an ordination in which the stands are color-coded by their current grazing status:

```
png("graphics/baseR.NMDS5.png",
    width = 4,
    height = 3,
    units = "in",
    pointsize = 10,
    res = 800)

plot(Oak1.z,
     display= "sites",
     type = "n",
     xaxt = "n",
     yaxt = "n",
     xlab = "",
     ylab = "")

points(Oak1.z$points[Oak$GrazCurr == "No", ],
```

```

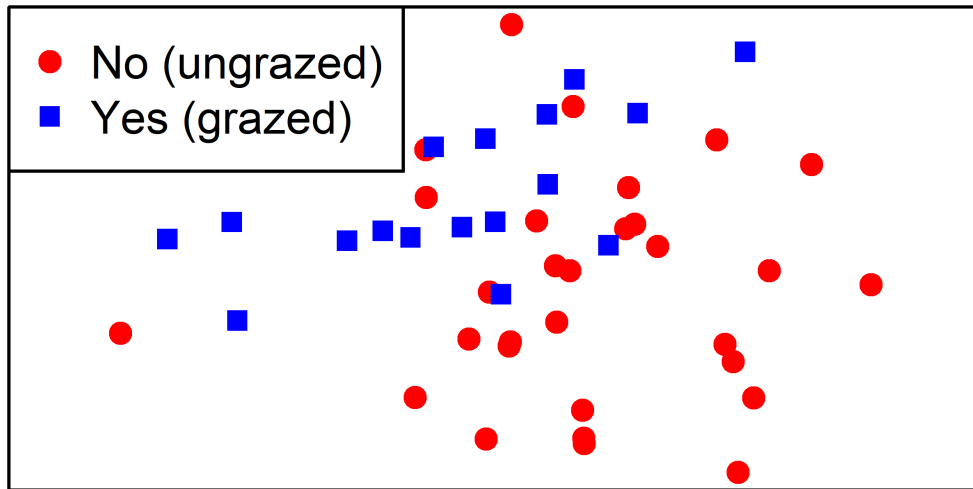
pch = 19,
col = "red")

points(Oak1.z$points[Oak$GrazCurr == "Yes",],
pch = 15,
col = "blue")

legend(x = "topleft",
pch = c(19,15),
col = c("red", "blue"),
legend = c("No (ungrazed)", "Yes (grazed)"))

dev.off()

```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by points, the color and shape of which indicate whether or not they experienced grazing when data were collected.

Saving Graphics in ggplot2 (ggplot2::ggsave())

The ggsave() function:

- Saves the last graph that was displayed to the specified filename.
- Recognizes the desired file format by the suffix provided in filename; options include .eps, .ps, .tex, .pdf, .jpeg, .tiff, .png, and .bmp.
- Allows you to control the:
 - Size of the image (width, height)
 - Units in which size is specified (units = c("in", "cm", "mm"))
 - Resolution of image in dots per inch (dpi).

For example, let's save an ordination in which the stands are color-coded by their current grazing status:

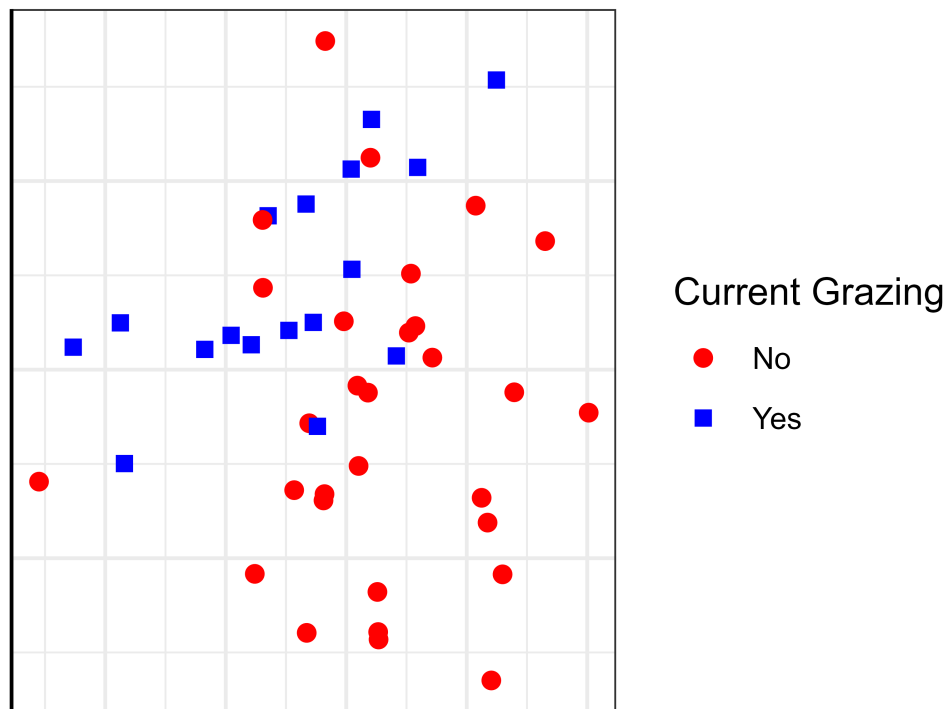
```

ggplot(data = Oak_explan,
aes(x = MDS1, y = MDS2)) +

```

```
geom_point(aes(shape = GrazCurr, color = GrazCurr),
size = 2)+
labs(shape = "Current Grazing",
color = "Current Grazing") +
scale_color_manual(values = c("red", "blue")) +
scale_shape_manual(values = c(19, 15)) +
theme.custom

ggsave(filename = "graphics/ggplot.NMDS6.png",
width = 4,
height = 3,
units = "in",
dpi = 800)
```



First two axes of a 3-dimensional NMDS ordination of the oak plant community dataset. Stands are shown by points, the color and shape of which indicate whether or not they experienced grazing when data were collected.

Conclusions

R is an incredibly powerful and flexible software for creating graphics based on the same data objects that are analyzed in it.

The base R plotting capabilities are helpful to know, but even more impressive graphics can be created easily via `ggplot2` and associated packages.

References

- Dalgaard, P. 2008. *Introductory statistics with R*. Second edition (First edition, 2002). Springer, New York, NY.
- Murrell, P. 2006. *R graphics*. Chapman & Hall/CRC, Boca Raton, LA.
- Sarkar, D. 2008. *Lattice: multivariate data visualization with R*. Springer, New York, NY.
- Venables, W.N., and B.D. Ripley. 2000. *Modern applied statistics with S*. Springer-Verlag, New York, NY.
- Wang, H. 2020. ggplot2 Theme Elements Demonstration. <https://henrywang.nl/ggplot2-theme-elements-demonstration/>
- Wickham, H. 2010. A layered grammar of graphics. *Journal of Computational and Graphical Statistics* 19(1):3-28. DOI:10.1198/jcgs.2009.07098

Media Attributions

- baseR.NMDS1
- baseR.NMDS2
- pch.symbols
- colors
- baseR.NMDS3
- baseR.NMDS4
- ggplot2.logo
- Wickham.2010_Figure1
- Wickham.2010_Figure2
- ggplot.NMDS1
- ggplot.NMDS2
- Wang_ggplot.themes
- ggplot.NMDS3
- ggplot.NMDS5
- baseR.NMDS5
- ggplot.NMDS6

43. Visualizing and Interpreting Ordinations

Learning Objectives

To understand how to interpret and customize ordination graphics.

To assess internal patterns by relating response variables to ordination axes and to one another in the ordination space.

To assess external patterns by considering the characteristics of the explanatory variables:

- Continuous variables can be shown as vectors and/or surfaces, and some ordinations can be rotated to maximize alignment with them
- Categorical variables can be shown as centroids, drawn with altered symbologies, and/or connected using spiders, hulls, and ellipses

Examples

```
require(vegan, tidyverse, ggvegan, ggordiplots)
```

Introduction

We've discussed a wide range of ordination techniques. Regardless of the technique used, it is important to understand some key principles for visualizing and interpreting ordinations. Here, I provide some general interpretation tips and then consider ways to delve into them, focusing on:

- Internal patterns – patterns from data that were part of the response matrix
- External patterns – relationships between an ordination and data that were not part of the response matrix. We often call these explanatory variables, but they may not be statistically related to the data. The ways that external patterns are explored differ with the characteristics of the variable:
 - Continuously distributed variables
 - Categorical variables

Note: This chapter focuses on two-dimensional graphs. Three-dimensional graphs are also available

through several packages in R – see the PCA chapter for examples. See also `vegan::ordisplom()` and `vegan::ordicloud()`.

Oak Example

We'll use the oak plant community dataset to illustrate graphical procedures. Use the `load.oak.data.R` script to load and make initial adjustments to the oak plant community dataset. The resulting object, `oak1`, contains abundances of the 103 most abundant species, each relativized by its maxima.

```
source("scripts/load.oak.data.R")
```

Use the `quick.metaMDS()` function to conduct a NMDS ordination of the `oak1` object for use below. To ensure repeatability of graphics, we'll set the random number generator immediately before doing so.

```
source("functions/quick.metaMDS.R")
```

```
set.seed(42)
```

```
Oak1.z <- quick.metaMDS(dataframe = Oak1,  
  dimensions = 3)
```

We're using a NMDS ordination here because of its flexibility, but many of the techniques discussed here can also be applied to other types of ordinations.

ggplot2 Extensions

The graphical approaches that are described below are available in base R, but many of them have also been 'translated' for use within the `ggplot2` 'universe'. This often requires re-organizing the data or automating the production of special types of graphics. I illustrate two such packages below.

Dozens of packages have been written that extend `ggplot2`. To see them, scroll through the list of packages available through CRAN that start with 'gg':

https://cran.r-project.org/web/packages/available_packages_by_name.html#available-packages-G

Some highlights that provide additional controls to existing `ggplot2` objects:

- `aplot` – aligns subplots to a main plot
- `cowplot` – aligns plots and arranges them into complex compound figures
- `ggalt` – extra coordinate systems, geoms, transformations, and scales
- `ggExtra` – adds marginal histograms/boxplots/density plots to scatterplots
- `ggiraph` – makes `ggplot` graphics interactive
- `ggpubr` – publication-ready plots
- `ggrepel` – automatically position non-overlapping text labels
- `ggsci` – color palettes used by various scientific journals (plus sci-fi movies, TV shows, etc.)
- `ggstar` – large range of symbol shapes
- `ggtea` – palettes and themes
- `ggthemes` – extra themes, scales, and geoms, including some to replicate the look of graphics by Edward Tufte

- `hrbrthemes` – extra themes, scales, and utilities, with an emphasis on typography
- `lemon` – tweaks for legends and axis lines, including in facets
- `tvthemes` – themes and palettes based on TV shows
- `xkcd` – creates graphs using the XKCD style

And, some packages that provide specific types of geometries:

- `gapmap` – gapped clustered heatmap: a heat map with associated dendrogram(s), but with gaps in the heatmap between the groups identified in the dendrogram
- `ggalluvial` – alluvial plots
- `ggasym` – asymmetric square matrix with different elements in upper and lower triangles and along diagonal
- `ggChernoff` – maps multivariate data as human-like faces (Chernoff 1973) – these are symbols in which elements (smile, brow, nose, eyes) can reflect particular variables. For example, the smile can range from a smile to a frown depending on the measured value for that sample unit. Manly & Navarro Alberto (2017) provide an example of this in their Figure 3.4.
- `ggdendro` – dendrograms and tree diagrams. See section about ‘Plotting a regression tree’ in the chapter about univariate regression trees for examples.
- `ggheatmap` – constructs heatmaps
- `ggmulti` – histograms and density functions for high dimensional data
- `ggord` –
- `ggordiplots` – `ggplot2` versions of `vegan`’s `ordiplot` functions
- `ggRandomForests` – classification and regression forests (aka random forests)
- `ggridges` – ridgeline plots showing changes in distributions over time or space
- `ggspectra` – for spectral and wavelength data
- `heatmaply` – interactive cluster heatmaps
- `tablesgg` – presentation-quality tables, displayed as plots

Additional packages are available through platforms like GitHub, as illustrated below.

The ggvegan Package

To use the object created from an ordination, you need to find and extract the required information (e.g., coordinates). Gavin Simpson has developed functions to simplify this by organizing objects created in `vegan` in a consistent manner so that they can be used in `ggplot2`. His approach is described here:

<https://github.com/gavinsimpson/ggvegan>

Simpson’s functions have been gathered together in a package called `ggvegan`. It is not currently available through CRAN, but can be installed by downloading the source code from GitHub.

```
install.packages("devtools")

devtools::install_github("gavinsimpson/ggvegan")

library(ggvegan)
```

The two key functions are `autoplot()` and `fortify()`. Each has methods that are automatically applied depending on the class of the object they are applied to. For example, if we applied `autoplot()` to an object of class ‘`metaMDS`’, the `autoplot.metaMDS()` function is called.

- The `autoplot()` function extracts the information (coordinates) and uses them to create a graph, but does not save them to a new object.
- The `fortify()` function summarizes the coordinates in a dataframe that can then be called

when creating a `ggplot2` object.

Let's save the coordinates in an object:

```
Oak1.z.gg <- fortify(Oak1.z)
```

Use `str()` to compare the information within `Oak1.z` and `Oak1.z.gg`.

Note that `fortify()` summarizes both sites scores and species scores (if the latter were requested). You therefore either have to filter the data to include the set of scores that you want. You can either do so while creating the object or during graphing. If we want to create an object that just contains the site scores:

```
Oak1.z.gg.sites <- fortify(Oak1.z) %>%  
  filter(score == "sites")
```

We can now call these coordinates in regular `ggplot2` functions.

The **ggordiplots** Package

The `ggordiplots` package makes `ggplot2` versions of `vegan`'s ordiplot functions (see 'Examining External Patterns with Categorical Variables' below for examples). It is available from CRAN:

```
install.packages("ggordiplots")
```

```
library(ggordiplots)
```

The primary function in this package is `gg_ordiplot()`. Ellipses, hulls, and spiders are each turned on (`TRUE`) or off (`FALSE`). I illustrate it below in the 'Examining External Patterns with Continuously Distributed Variables' section.

Additional functions include:

- `gg_envfit()`
- `gg_ordisurf()`
- `gg_ordibubble()`
- `gg_ordicluster()`

General Interpretation Tips for Ordinations

Interpretation of ordinations can focus on the axes and/or the points.

Axes:

- Many techniques automatically result in uncorrelated (orthogonal) axes. However, those axes are not important for all ordinations.
- Numerical scales on axes may be useful for PCA. It is often useful to add interpretive aids to the axes, drawing on the meaning that you assign to it based on the loadings. For example:

- Arrows to show that plant size increases as you move in one direction, or that points arrayed to one side tend to drier and those arrayed to the other side tend to be wetter.
- Add line drawings of the two extremes on either end of the axis.
- In eigen-based techniques (PCA, PCoA), the order of the axes is important. In particular, the axes are always in descending order of importance. However, this does not mean that you can only view the first two axes. For example, it is perfectly appropriate to plot PC1 vs. PC3.
- When axes matter, indicate which axes are being graphed and their importance (% of variance explained).
- Numerical scales on axes are generally not useful for NMDS. To avoid axes being over-interpreted, we often do not even show them in a NMDS.

The point cloud:

- Direction of axes (left vs right, up vs down) is often arbitrary. This may vary among runs – for example, if the signs of the loadings in a PCA are reversed between two runs. It is therefore important to generate the loadings, graphics and any interpretive aids in the same run.
- By default, NMDS ordinations are organized to show as much variation as possible – recall that one of the last steps in `metaMDS()` was to use PCA to rotate the final coordinates of the NMDS solution. However, this does not mean that you can only view the point cloud from this perspective.
- NMDS ordinations can be rotated to focus on any aspect of the ordination cloud **without affecting its interpretation**. See examples below in the 'Examining External Patterns with Continuously Distributed Variables' section.
- Ordinations can be reflected (to mirror image) to emphasize particular patterns in the data. For example, if you have two ordinations of data from the same sample units (e.g., Fig. 19.2 and 19.3 in McCune & Grace 2002), interpretation will be enhanced if groups of sample units are in the same relative position in both graphs.
- Orthogonal rotations (ch. 15 of McCune & Grace 2002) can be used to view NMDS data clouds from another angle. These rotations change the correlation of variables with ordination axes. However, they do not change the cumulative variance explained by the ordination. Varimax rotation is the most common method, though many other rotation methods can also be used. Since rotations and other adjustments affect the correlations of environmental variables with axes, all such adjustments must be completed before you begin to overlay other variables, etc.

Examining Internal Patterns

By internal patterns, I mean patterns from data that were incorporated into the ordination itself.

How well does an ordination 'fit' the data? Eigenanalysis techniques (PCA, CA, PCoA) report eigenvalues that correspond to the variation explained or represented by each axis. A more generally applicable technique is to compare the correlations between distances in the original data and distances in ordination space. A Shepard plot is an example of this kind of analysis. McCune & Grace (2002, p. 112) suggest as a rule-of-thumb that an ordination that explains 30-50% of the variation in two axes may be useful and interpretable while an ordination that explains > 50% of the variation is very good. This depends, of course, on sample size, the characteristics of the data, and other considerations.

Correlations of Species Abundances with Ordination Axes

We can also investigate individual variables (i.e., columns in the response matrix). These can also be assessed with techniques like Indicator Species Analysis and TITAN. For simplicity, I'll focus on the

```
trees <- Oak_spp %>%
  filter(GrowthForm == "Tree") %>%
  pull(SpeciesCode)

tree.data <- Oak1 %>%
  select(any of(trees))
```

```
cor(tree.data, Oak1.z$points) %>%
  round(2)
```

	MDS1	MDS2	MDS3
Abgr.t	0.29	-0.30	-0.26
Acma.t	0.28	-0.20	-0.35
Conu.t	0.05	-0.04	0.16
Frla.t	-0.27	-0.23	-0.11
Prav.t	0.26	0.03	0.22
Psme.t	0.25	-0.35	0.07
Pyco.t	-0.27	0.05	-0.07
Quga.t	0.10	0.25	-0.05
Rhpu.t	0.38	-0.11	-0.29

Species Abundances Along Ordination Axes

```
vegemite(x = tree.data,  
  use = Oakl.z$points[, "MDS1"],  
  scale = "Domin",  
  zero = "-")
```

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS  
ttttttttttttttttttttttttttttttttttttttttttttttttttttttttt  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
```

```

                ddddddddddddddddddddddddddddddddddddddddddddddd
23443004301033010304121120411242423342231321201
36313614922307807546664109753452071821527498859
Fr1a.t --3--3-----3-----
Pyco.t -1312-----1-2---2-----1-----2---1---
Quga.t 3222223332242232232222232222333222222232322222
Conu.t --2--2-----3-----2
Psme.t --1-----2--1---1---11--22-----2222-2-1-3--2-
Acma.t --2--1---11-----1--1-22-1-12---212222-----23-
Prav.t -----1-----2-----1-1-----13--12--2--1--2
Abgr.t -----3-----3-----2--3-
Rhpu.t -----1-----3-----2-----323
47 sites, 9 species
scale: Domin

```

The top several rows of this output are the stand numbers, printed compactly and vertically – stand 23 is on the left and stand 19 is on the right. The stands themselves are ordered by their first axis scores:

```

sort(Oak1.z$points[,1]) %>%
  round(2)

```

```

Stand23 Stand36 Stand43 Stand41 Stand33 Stand06 Stand01 Stand44
-1.27 -1.13 -0.94 -0.92 -0.59 -0.48 -0.39 -0.38
Stand39 Stand02 Stand12 Stand03 Stand30 Stand37 Stand08 Stand10
-0.35 -0.35 -0.32 -0.24 -0.22 -0.17 -0.16 -0.15
Stand07 Stand35 Stand04 Stand46 Stand16 Stand26 Stand14 Stand11
-0.14 -0.12 -0.09 -0.09 -0.09 -0.01 0.02 0.02
Stand20 Stand09 Stand47 Stand15 Stand13 Stand24 Stand45 Stand22
0.05 0.05 0.09 0.10 0.11 0.13 0.13 0.13
Stand40 Stand27 Stand31 Stand38 Stand42 Stand21 Stand25 Stand32
0.21 0.26 0.27 0.29 0.29 0.36 0.54 0.56
Stand17 Stand34 Stand29 Stand18 Stand28 Stand05 Stand19
0.59 0.60 0.62 0.65 0.70 0.82 1.01

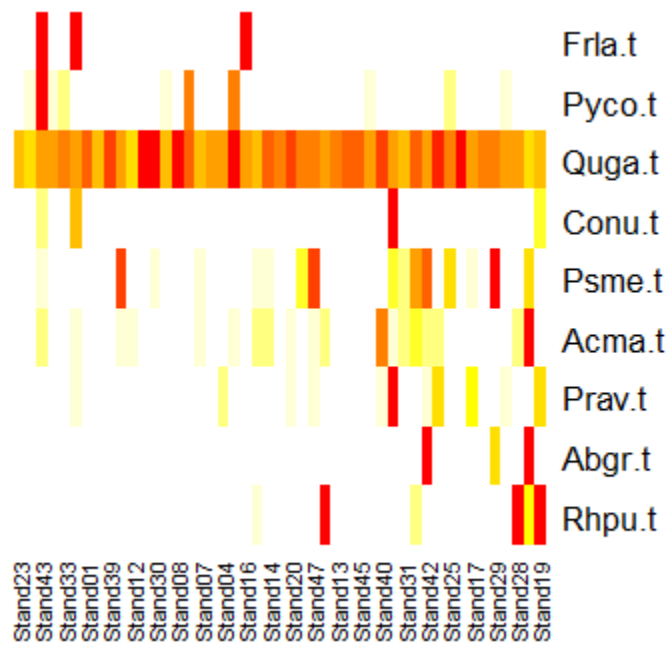
```

A visual variant of this is `tabasco()`, which uses a heat map to show the magnitude of each variable. We applied this previously to a dendrogram from an hierarchical cluster analysis, but it can also be applied to ordination axes, continuously distributed external variables, categorical external variables, and other types of data. Let's view the abundance of each tree species relative to the first axis of the NMDS ordination:

```

tabasco(x = tree.data,
  use = Oak1.z$points[, "MDS1"])

```



Heat map of showing the relative abundance of each tree species within the oak stands. Stands are organized based on their position along the first NMDS ordination axis.

Many trees occur infrequently. The species Frla.t tends to occur in stands with small values on the first ordination axis (i.e., if these coordinates were graphed, it would be in some of the plots on the left side of the ordination). In contrast, species like Abgr.t and Rhpu.t tend to occur in stands with high values on the first ordination axis – if graphed, they would be in some of the plots on the right side of the ordination.

The abundance patterns of each species can be correlated with ordination axes using the `vegan::envfit()` function.

```
envfit(Oak1.z ~ .,
  data = tree.data,
  permutations = 0)
```

***VECTORS

	NMDS1	NMDS2	r2
Abgr.t	0.54828	-0.83629	0.1725
Acma.t	0.68050	-0.73275	0.1175
Conu.t	0.62233	-0.78276	0.0040
Frla.t	-0.61693	-0.78702	0.1276
Prav.t	0.98336	0.18165	0.0700
Psme.t	0.43699	-0.89947	0.1875
Pyco.t	-0.96807	0.25069	0.0768
Quga.t	0.25908	0.96586	0.0705
Rhpu.t	0.91547	-0.40239	0.1562

These values can also be used to draw vectors on an ordination, just like we've seen with PCA biplots. See the 'Continuously Distributed External Variables' section below for an example of this and some caveats about the usage of `envfit()`.

External Patterns

External variables are those that were not part of the response matrix to which an ordination was applied. These might be variables that were not considered during the ordination, or those that were included as constraints during a direct gradient analysis (RDA, dbRDA).

How we visualize external variables depends on whether they are continuously distributed or categorical. Each type of variables is dealt with separately below.

Which External Variables to Focus on?

Generally we would focus our attention on those variables that are statistically significantly related to the response distance matrix. We could visualize patterns related to an external variable that is not statistically related to the distance matrix, though the patterns may be less helpful and we would need to remember when interpreting them that the relationship is not significant.

Some of the functions described below can conduct statistical tests of the fit between an external variable and the locations of the sample units in ordination space. It is important to think carefully about whether these statistical comparisons are necessary or helpful. In particular, **the coordinates of the sample units in ordination space are an imperfect representation of the actual data on which the ordination was based**. For example:

- The first few PCs from a PCA only capture a subset of the variance in the data
- The coordinates in a NMDS do not perfectly mirror the patterns in the original distance matrix – see the discussion of stress in the NMDS chapter if this does not make sense.

As a result, a statistical test based on the coordinates is an **approximation** of the actual relationship between the explanatory variable and the distance matrix derived from the sample units. **Direct** tests of the relationship between an explanatory variable and the distance matrix are accomplished by statistical techniques such as PERMANOVA.

Constrained ordinations such as RDA and dbRDA provide a stronger connection between the original data and the ordination coordinates. However, it is important to remember that a constrained ordination behaves in an unconstrained manner as the number of constraining variables (continuous variables or levels of a categorical variable) increases. Also, if an ordination has three or more constrained axes (e.g., if three continuous variables were tested, or if one categorical variable with four levels was tested), then a two-dimensional ordination is still only showing a subset of the explained variance. See the 'Comparing Ordination Techniques' chapter for an example of the connection between NMDS and dbRDA.

Continuously Distributed External Variables

Continuous variables (e.g., pH, elevation, etc.) can be plotted onto an ordination and the correlation with each axis quantified. Symbol sizes can also be made proportional to data values, as illustrated in the NMDS notes. Other common ways to explore these patterns are through vectors and surfaces.

vegan::envfit() for Vectors

A PCA biplot includes a vector representing each variable. One way to add such vectors ourselves is via the `vegan::envfit()` function. This function can be applied to multiple variables simultaneously and identifies, for each variable, the direction in ordination space in which it changes most rapidly and has maximal correlation with the ordination configuration. It can be applied to variables that were part of the response (as in PCA) and to variables that were not part of the ordination, which is our focus here.

By default, `envfit()` conducts permutation-based statistical tests of the fit between each variable and the ordination axes. However, I recommend not using these tests for a few reasons:

- As noted in ‘External Patterns’ above, tests between a variable and the representation of a distance matrix in ordination space are not the same as direct tests between a variable and a distance matrix.
- When we are interested in the fit of a variable with an ordination, we’re generally more interested in its overall fit than in its fit with individual axes.

Statistical tests can be turned off by setting the number of permutations to zero.

To illustrate this function, let’s identify the direction of greatest change for the geographic variables of each stand compared to their locations in the Oak1 NMDS ordination:

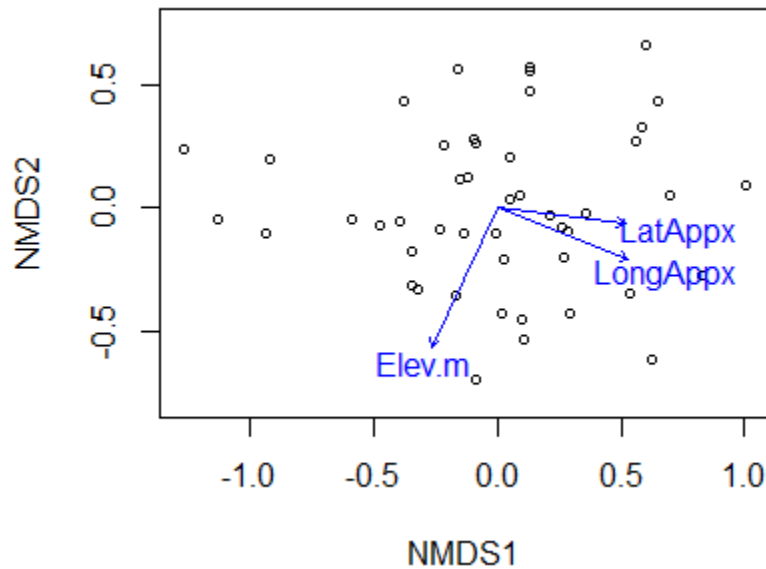
```
geog.fit <- envfit(Oak1.z ~ Elev.m + LatAppx + LongAppx,  
  data = Oak,  
  permutations = 0)
```

```
geog.fit
```

```
***VECTORS  
  
      NMDS1    NMDS2    r2  
Elev.m  -0.43216 -0.90180 0.0620  
LatAppx   0.99236 -0.12338 0.0428  
LongAppx  0.93106 -0.36488 0.0513
```

The resulting vector(s) can be overlaid onto the ordination plot:

```
plot(Oak1.z,  
  display= "sites")  
  
plot(geog.fit)
```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols. The vectors show the directions in which latitude, longitude, and elevation increase most strongly.

Note that we have to draw the ordination first, and then add the vectors onto it. In this case, I have not hidden the axis labels, values, and tick marks though as previously discussed it is common to do so for NMDS ordinations.

In the PCA notes, we discussed principles for interpreting vectors. Those principles are also applicable here. For example:

- The angle between two vectors indicates the direction of the relationship between them:
 - Narrow = positive correlation
 - Perpendicular = uncorrelated
 - Opposing = negative correlation
- The length of a vector often indicates the strength of the relationship. However, length can be calculated several different ways and it isn't always clear if/how vectors have been scaled. Borcard et al. (2018) discuss this in detail.
- The relative location of a sample unit or species along a vector is found by extending a perpendicular line from the vector to the data point.

Note: you can always draw a vector on an ordination, but that does not mean that it is meaningful. For example, the relationship between the variable and the ordination space may not be linear. If the response is non-linear, knowing the direction in which it increases most strongly may be of limited utility. Surfaces (see below) are a more useful way to reflect these relationships.

Rotating An Ordination

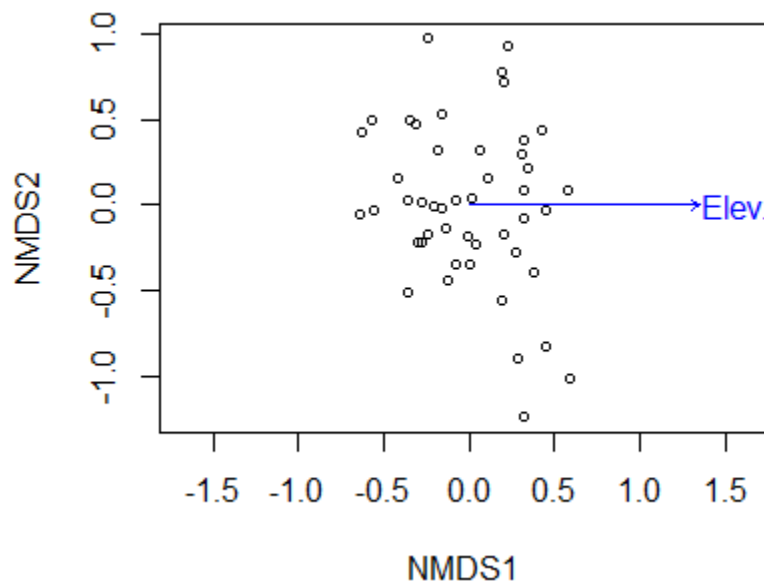
Rather than just overlaying a variable onto an NMDS ordination, it is often helpful to rotate the ordination so that it aligns with one (or more) of the variables. One way to do this is via the `vegan::MDSrotate()` function. Let's use it to rotate the ordination with respect to elevation:

```
Oak1.z.elev <- MDSrotate(Oak1.z, Oak$Elev.m)
```

The rotated coordinates have the same names as in the original object and can be extracted and used in `ggplot2` graphics, etc.

We can verify the rotation by plotting the rotated coordinates and then superimposing the vector for elevation onto it:

```
plot(Oak1.z.elev,  
     display = "sites")  
  
plot(envfit(Oak1.z.elev ~ Elev.m,  
            data = Oak,  
            permutations = 0))
```



Two axes of a three-dimensional NMDS ordination of the oak plant community, after rotating so that the first axis aligns as strongly as possible with elevation. Stands are indicated by open symbols.

We have now rotated the point cloud so that elevation is perfectly correlated with NMDS1. The correlation between individual species and this axis is now the same as the correlation between the representation of those species in ordination space and elevation – it is not the correlation between those species and elevation.

Note that a rotation like this is perfectly appropriate for ordinations like NMDS and PCoA where the

axes have little meaning, but would be inappropriate for ordinations like PCA where the axes have particular meanings.

Surfaces

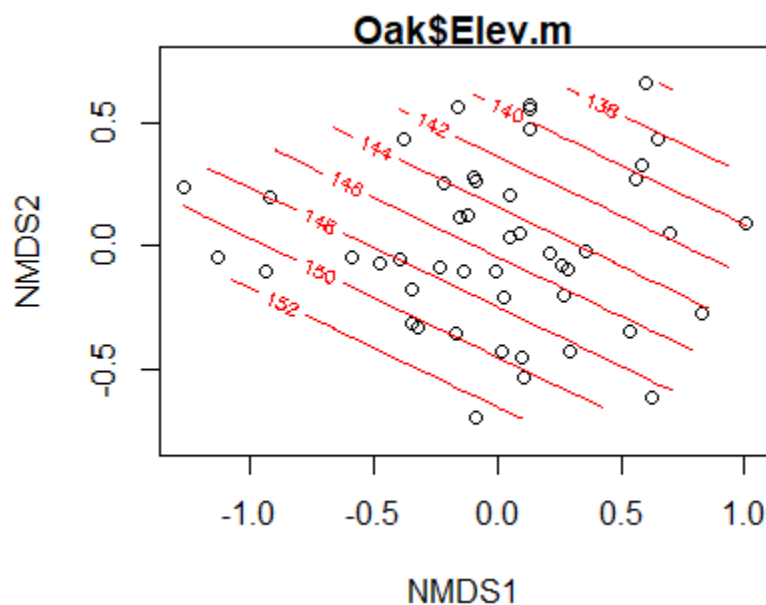
If your interest is in a single variable, or several variables individually, it can be helpful to map them as surfaces. A surface is like a contour map, and allows you to examine whether the patterns are linear or non-linear.

`vegan::ordisurf()`

The `vegan::ordisurf()` function is one way to create a surface. This function fits a generalized additive model (GAM) predicting the variable as a function of the scores on the ordination axes; see Simpson (2011) for an explanation and instructions on how you can further customize the surface calculation.

For example, let's apply this to elevation:

```
ordisurf(x = Oak1.z,  
         y = Oak$Elev.m)
```

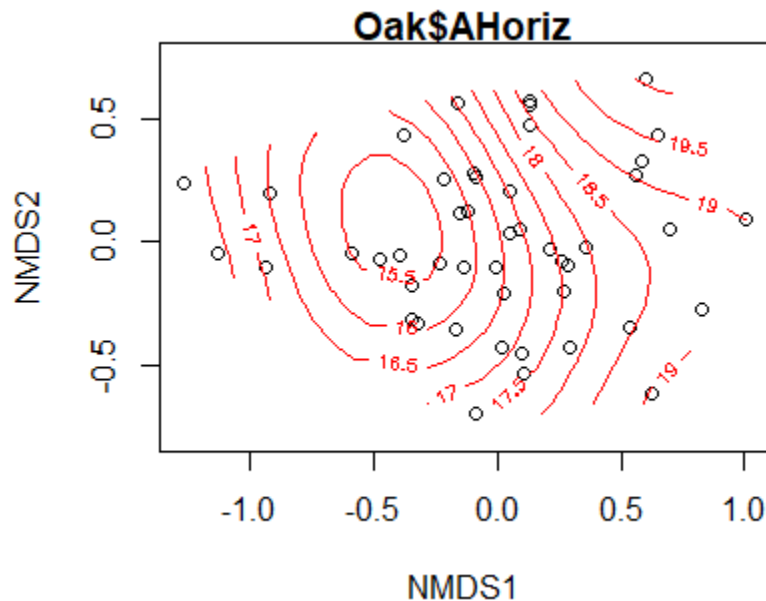


First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols. The contours connect points in the ordination space that are predicted to have the same elevation.

The parallel contours here suggest that there is a linear relationship between elevation and the locations of stands in ordination space. How would these contours look if we applied this to the coordinates that we rotated with respect to elevation (`Oak1.z.elev`).

What about the relationship with the depth of the A horizon?

```
ordisurf(x = Oak1.z,  
y = Oak$AHoriz)
```

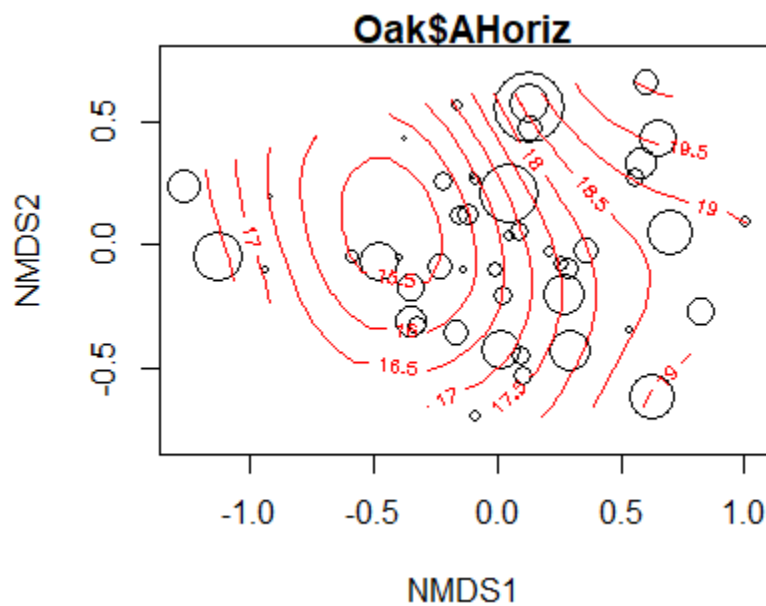


First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols. The contours connect points in the ordination space that are predicted to have the same depth of the A horizon.

The relationship here is clearly not linear – the lowest values are predicted to be just to the left of center of the point cloud, with values increasing in all directions.

A surface can be produced for any continuous variable, but that doesn't mean it fits the data well. One way to assess its fit is to make the size of the points proportional to their value of the explanatory variable. The `bubble` argument does this; we simply need to specify the maximum symbol size to be used. While doing so, I'll also adjust some of the other arguments.

```
ordisurf(x = Oak1.z,  
y = Oak$AHoriz,  
bubble = 5)
```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols, the size of which is positively related to the depth of their A horizon. The contours connect points in the ordination space that are predicted to have the same depth of the A horizon.

The fitted values of the explanatory variable can be obtained using the `calibrate()` function:

```
calibrate(ordisurf(Oak1.z,  
Oak$AHoriz))
```

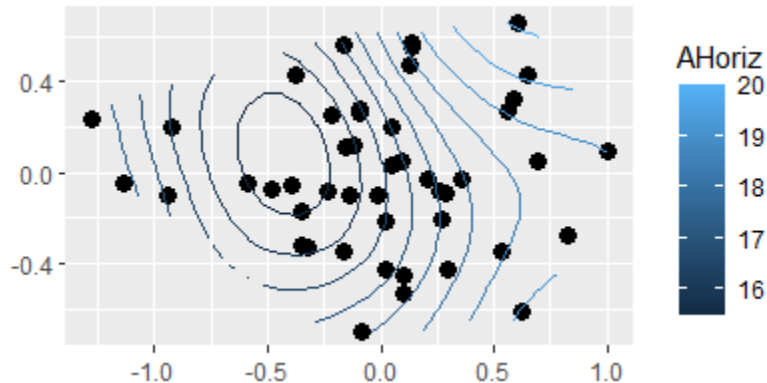
1	2	3	4	5	6	7	8	9	10
15.26	15.48	15.50	16.45	18.85	15.32	15.82	17.01	17.07	15.81
11	12	13	14	15	16	17	18	19	20
16.52	15.89	17.38	16.86	17.18	17.42	19.22	19.58	19.00	16.70
21	22	23	24	25	26	27	28	29	30
17.97	18.71	17.87	18.38	18.46	16.33	17.56	18.72	19.03	15.81
31	32	33	34	35	36	37	38	39	40
17.54	19.05	15.48	20.05	15.98	17.67	16.18	17.65	15.86	17.39
41	42	43	44	45	46	47			
16.46	17.82	16.91	15.78	18.66	16.43	16.94			

These values could be saved and used to calculate the residual for each sample unit, which could also be plotted or examined.

ggordiplots::gg_ordisurf()

To fit a surface with `ggordiplots`:

```
gg_ordisurf(ord = Oak1.z,
  env.var = Oak$AHoriz,
  var.label = "AHoriz",
  binwidth = 0.5)
```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols. The contours connect points in the ordination space that are predicted to have the same depth of the A horizon.

Other Approaches

Some continuously distributed explanatory variables are not well illustrated by vectors or surfaces. For example, suppose your data are from re-measurements of permanent plots. A very helpful way to explore the effect of time might be to connect successive measurements of a given plot with an arrow. These successional vectors can be drawn using the function `vegan::ordiarrows()`; see its help file for details. Davies et al. (2012) provide an example of this approach.

Categorical External Variables

Categorical variables such as grazing history (yes / no) or drainage class (well / good / moderate / poor) can be examined by considering where the data points associated with each level of a variable occur. This type of display generally focuses on the centroids of each level and/or the perimeter defined by the group.

vegan::envfit() for Centroids

The function `envfit()` that we used above can also be used to determine and compare the centroids of groups. It deals with variables appropriately based on their class. For categorical variables, it calculates the mean score for each group on each axis of the ordination. As above, we can omit permutation tests of statistical significance by requesting zero permutations.

```
envfit(Oak1.z ~ GrazCurr,
  data = Oak,
  permutations = 0)
```

```

***FACTORS:

Centroids:
      NMDS1    NMDS2
GrazCurrNo  0.1386  0.1032
GrazCurrYes -0.2447 -0.1821

Goodness of fit:
      r2
GrazCurr 0.1555

```

Interpretation of a statistical test here would be subject to the same concerns highlighted above for continuous variables.

Plotting this result is not very interesting (try it!).

Considering Continuous and Categorical Variables Simultaneously

Since `envfit()` recognizes the class of variables and analyzes them appropriately, it can be used to simultaneously examine both types of variables. For example:

```

envfit(Oak1.z ~ Elev.m + GrazCurr,
data = Oak,
permutations = 0)

```

```

***VECTORS

      NMDS1    NMDS2    r2
Elev.m -0.43216 -0.90180 0.062

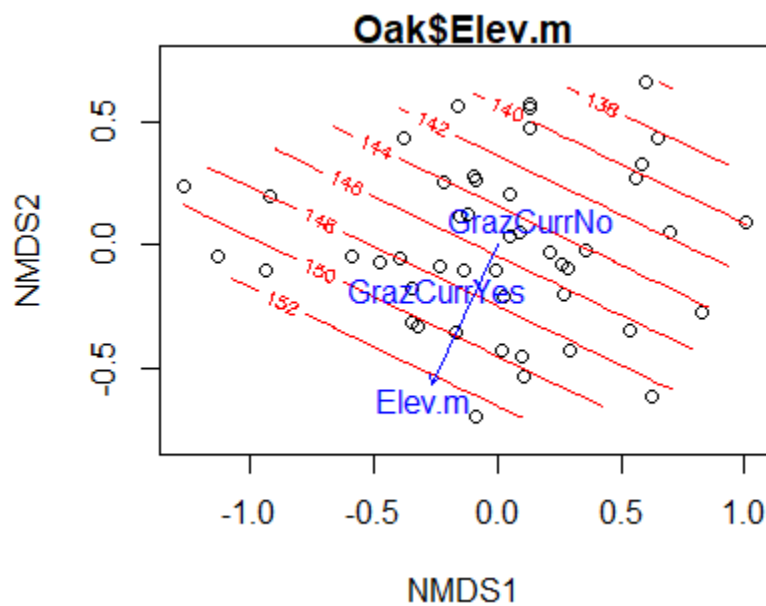
***FACTORS:

Centroids:
      NMDS1    NMDS2
GrazCurrNo  0.1386  0.1032
GrazCurrYes -0.2447 -0.1821

Goodness of fit:
      r2
GrazCurr 0.1555

```

Can you figure out how I created this graph?



First two axes of a three-dimensional NMDS ordination of the oak plant community. Stands are indicated by open symbols. The contours connect points in the ordination space that are predicted to have the same elevation. The elevation vector points in the direction in which it increases most strongly. The 'GrazCurrYes' and 'GrazCurrNo' labels indicate the centroids of the stands within each group.

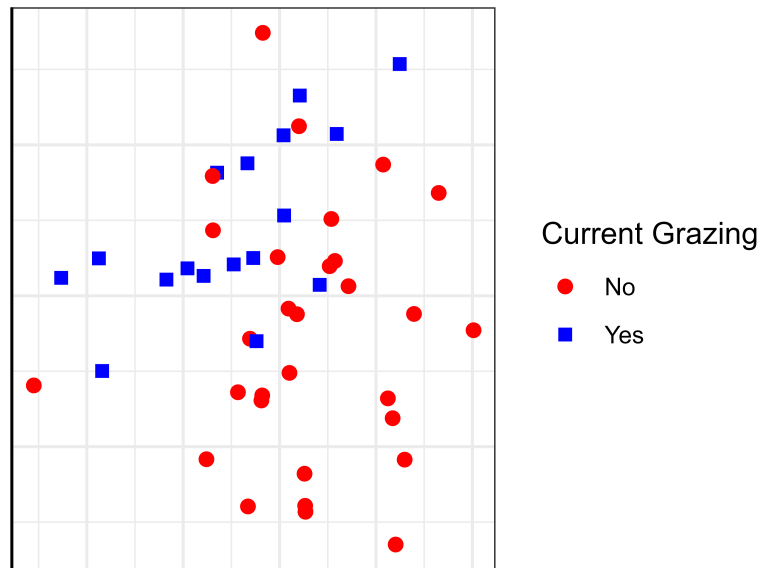
Symbology

A more interesting way to plot differences among groups is to use a different symbol for each group.

In base R graphing, this requires indexing to select different subsets of the data for each type of symbol. We will often also control other aspects of the figure while doing this. For example, we might plot the two grazing classes in different symbols and colors, and add a legend defining them.

In `ggplot2`, symbology is much more easily controlled.

See the examples in the 'General Graphing Principles' chapter for examples of how both types of graphics can be customized. The resulting `ggplot2` image is repeated here:



First two axes of a three-dimensional NMDS ordination of the oak plant community. Symbol shape and color indicate whether or not stands were grazed at the time of data collection.

Spiders, Ellipses, Hulls, etc.

The `vegan` package contains several functions designed specifically to visualize the relationship between ordinations and categorical explanatory variables:

Function	Display Details
<code>ordispider()</code>	Connects each data point to the centroid for the group
<code>ordiellipse()</code>	Draws an ellipse that encompasses the specified standard deviation or standard error to a specified confidence limit, or an ellipsoid hull that encloses all points
<code>ordibar()</code>	Draws a cross showing the standard deviations or standard errors that are also used to locate the ellipse
<code>ordihull()</code>	Connects the outermost points in each level.
<code>ordicluster()</code>	Overlays dendrogram onto ordination (see the 'Using Groups' chapter for details).

In base R, each of these functions could be plotted separately or overlain on a graph like the one we created above. To do so, just run the desired function after creating the graph. Depending on whether functions are run before or after plotting the points, the lines will appear either under or above the points.

`vegan::ordispider()`

For example:


```

plot(Oak1.z,
     display= "sites",
     type = "n",
     xaxt = "n",
     yaxt = "n",
     xlab = "",
     ylab = "")

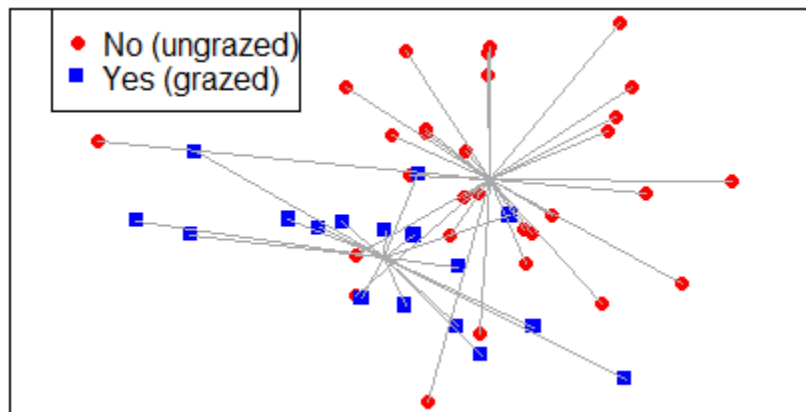
points(Oak1.z$points[Oak$GrazCurr == "No",],
       pch = 19,
       col = "red")

points(Oak1.z$points[Oak$GrazCurr == "Yes",],
       pch = 15,
       col = "blue")

legend(x = "topleft",
       pch = c(19,15),
       col = c("red", "blue"),
       legend = c("No (ungrazed)", "Yes (grazed)"))

ordispider(Oak1.z,
           Oak$GrazCurr,
           col = "dark grey")

```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Symbol shape and color indicate whether or not stands were grazed at the time of data collection. The grey lines connect stands to the centroid of their group.

ggordiplots::gg_ordiplot()

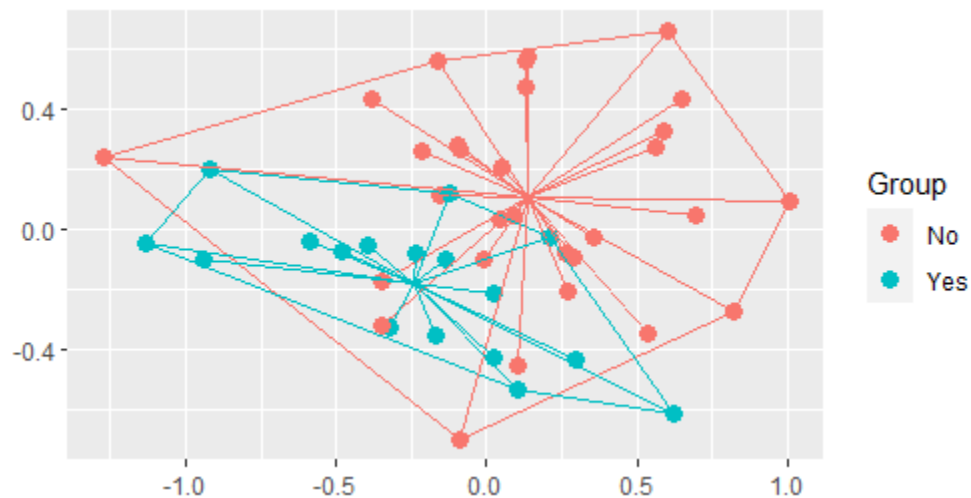
With `ggordiplots`, we can use a single function – for example, let's add spiders and hulls but not ellipses:

```

gg_ordiplot(ord = Oak1.z,
            groups = Oak$GrazCurr,
            ellipse = FALSE,

```

```
hull = TRUE,  
spiders = TRUE)
```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Symbol color indicates whether or not stands were grazed at the time of data collection. Lines connect stands to the centroid of their group and encompass all stands within the same group.

Other Approaches

Many other options are possible. For example, in `ggplot2` you can use faceting to graph each level of a categorical variable separately. In this situation, you can either graph the sample units from each level only in the relevant level, or you can show them in all levels (e.g., as small grey symbols) and highlight (larger, colored symbols) the relevant sample units in a given level.

Conclusions

I've stressed that ordinations primarily are data reduction and visualization tools. The considerations in this chapter primarily focus on visualizations. These visualizations are often most appropriately thought of as supporting a statistical analysis. Once you've identified statistically significant patterns through statistical techniques such as PERMANOVA, use ordinations to illustrate those patterns.

Ordinations are a key tool for communicating your conclusions, and are also an opportunity to create memorable and easily understood graphics.

References

Chernoff, H. 1973. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association* 68(342):361–368.

Davies, G.M., J.D. Bakker, E. Dettweiler-Robinson, P.W. Dunwiddie, S.A. Hall, J. Downs, and J. Evans. 2012. Trajectories of change in sagebrush steppe vegetation communities in relation to multiple wildfires. *Ecological Applications* 22:1562–1577.

Manly, B.F.J., and J.A. Navarro Alberto. 2017. *Multivariate statistical methods: a primer*. Fourth edition. CRC Press, Boca Raton, FL.

McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.

Simpson, G. 2011. What is ordisurf() doing...? <http://www.fromthebottomoftheheap.net/2011/06/10/what-is-ordisurf-doing/>

PS: vegan Hexagon Logo

Packages in the `tidyverse` and elsewhere have hexagon logos. I found this on GitHub and thought you might be interested. 😊



This image was created by Zane Dax and posted to <https://github.com/vegandevs/vegan/issues/436>.

Media Attributions

- `tabasco.tree`
- `geog.fit1`
- `geog.fit2`
- `ordisurf.elev`
- `ordisurf.Ahoriz`
- `ordisurf.Ahoriz2`
- `ggordisurf.Ahoriz`
- `ordisurf.envfit`
- `ggplot.NMDS6`
- `ordispider`

- `ggordiplot.spider`
- `vegan.logo`

PART V

FOLLOW-UP TESTS

Introduction

After covering some foundational concepts, we covered topics that fall into three main categories:

- Testing for differences among *a priori* groups
- Identifying groups within your data (classification)
- Visualizing multivariate data

For example, you might use PERMANOVA to identify a difference in composition between levels of a factor, or you might use a hierarchical cluster analysis to identify groups of plots that are similar to one another in terms of their distances. After applying either technique, you might then conduct an ordination to show the overall patterns across the set of response variables.

However, what else can you do to figure out what the multivariate patterns mean?

A Multivariate Test as an ‘Insurance Policy’

If you conduct enough statistical tests, you will find something statistically significant. A multivariate test can serve as an insurance policy that reduces the chances of such spurious patterns. Recall that we are testing the multivariate set of responses simultaneously. **If a multivariate test is not significant, there is no justification to test each univariate response.**

A statistically significant multivariate test is an indication that there are patterns to be explored and understood. However, multivariate tests don't show you directly which response variable(s) are driving the patterns. For example, patterns could be the net effect of small changes in many response variables, or could reflect large changes in a small number of response variables. Furthermore, some response variables may be affected by one explanatory variable while others are affected by a different explanatory variable.

Decisions about how to analyze individual response variables should incorporate the characteristics of the data. In this context, the key distinction is between non-compositional and compositional data.

Non-Compositional Data

Non-compositional data are generally continuously distributed and have few zeroes – each variable has a measured value in each sample unit.

If response variables are strongly correlated with one another, it is often helpful to capitalize on the fact that eigen-based techniques create new synthetic variables that are uncorrelated with each other. For example, although every principal component (PC) is a linear combination of the original variables, by definition they are uncorrelated with the other PCs from that eigenanalysis. This means that each PC can be interpreted separately and analyzed independently.

Continuously-distributed response variables that are not strongly correlated with one another are

often analyzed using a separate linear model for each variable. Furthermore, techniques like PERMANOVA and RRPP can be applied identically to both multivariate and univariate data, greatly simplifying follow-up analyses – the response changes but the structure of the model does not change.

Compositional Data

Compositional data (e.g., abundances of taxa within communities) are different from many other types of data. In particular, the matrix is often sparse (i.e., contains many zeroes). These absences are part of the patterns exhibited by individual taxa. If we simply calculate an average abundance for each taxon in each group, we are potentially ignoring a lot of information about their distributions.

The chapters in this part survey several techniques to identify the species (response variables) associated with the observed patterns:

- Similarity percentage (SIMPER) – to identify the relative importance of each species in distinguishing between two levels of a categorical variable.
- Indicator Species Analysis (ISA) – to identify species that are strongly associated with (i.e., indicative of) one level or a set of levels of a categorical explanatory variable.
- Threshold Indicator Taxa ANalysis (TITAN) – to identify species that are strongly associated with low or high values of a continuous explanatory variable.

Other techniques are also possible – for example, Hu & Satten (2020) proposed a linear decomposition model that can both test for overall effects of a treatment and test whether individual taxa differ with respect to that treatment. The software to do so is provided in their `LDM` package, and includes an alternate implementation of PERMANOVA.

Oak Example

We will use the oak plant community dataset to illustrate these follow-up tests. Use the `load.oak.data.R` script to load and make initial adjustments to the oak plant community dataset.

```
source("scripts/load.oak.data.R")
```

The resulting object, `oak1`, contains abundances of the 103 most abundant species, each relativized by its maxima.

Categorical Explanatory Variable (Grazing)

To illustrate the ideas below, we'll use an explanatory variable that identifies the combined grazing status – past and current – of each stand. There are three combinations of these grazing statuses. We used these combinations in previous notes, but the levels had unhelpful names (e.g., 'No_No'); let's give them more intuitive names this time.

```
Oak_explan <- Oak_explan %>%
  rownames_to_column(var = "Stand") %>%
  rowid_to_column(var = "ID") %>%
  mutate(GP_GC = paste(GrazPast, GrazCurr, sep = "_")) %>%
  merge(y = data.frame(GP_GC = c("No_No", "Yes_No", "Yes_Yes")),
```

```
Grazing = c("Never", "Past", "Always")) %>%
  arrange(ID)
```

Note that we end this code by arranging the rows by ID to ensure that they remain in the same order as they are in our response matrix (Oak1).

```
summary(as.factor(Oak_explan$Grazing))
```

```
Always  Never  Past
      17     24     6
```

Does composition differ among grazing statuses? We can test this via PERMANOVA:

```
set.seed(42)
```

```
adonis2(Oak1 ~ Grazing,
  data = Oak_explan,
  distance = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ Grazing, data = Oak_explan, distance = "bray")
      Df SumOfSqs      R2      F Pr(>F)
Grazing  2   0.9095 0.07844 1.8725 0.006 **
Residual 44  10.6854 0.92156
Total    46  11.5949 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Grazing treatments do differ in composition.

Let's also visualize composition patterns and grazing treatments. We'll use NMDS here so that we see the grazing treatments in the context of all of the compositional variation expressed in these dimensions:

```
source("functions/quick.metaMDS.R")
```

```
set.seed(42)
```

```
Oak.nmnds <- quick.metaMDS(Oak1, 3)
```

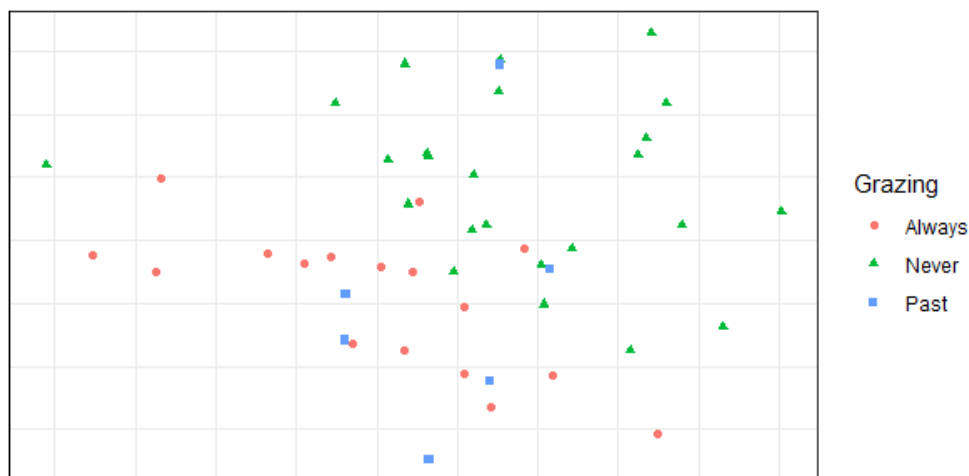
```
library(ggvegan)
```

```
Oak.nmnds.gg <- fortify(Oak.nmnds) %>%
  filter(score == "sites")
```

```
Oak_explan <- Oak_explan %>%
  merge(y = Oak.nmnds.gg, by.x = "Stand", by.y = "label")
```

```
theme.custom <- theme_bw() +
  theme(axis.line = element_line(),
        axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank())

ggplot(data = Oak_explan, aes(x = NMDS1, y = NMDS2)) +
  geom_point(aes(colour = Grazing, shape = Grazing)) +
  theme.custom
```



First two axes of a three-dimensional NMDS ordination of the oak plant community. Symbol shape and color distinguish groups based on their grazing history.

We'll use SIMPER and ISA to explore which species are associated with these groups.

Continuously-Distributed Explanatory Variable (PDIR)

How does composition vary with light availability (PDIR)? We can also analyze this with PERMANOVA:

```
set.seed(42)

adonis2(Oak1 ~ PDIR,
  data = Oak_explan,
  distance = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ PDIR, data = Oak_explan, distance = "bray")
      Df SumOfSqs      R2      F Pr(>F)
```



```

PDIR      1    0.8068 0.06958 3.3655  0.002 **
Residual 45    10.7881 0.93042
Total    46    11.5949 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

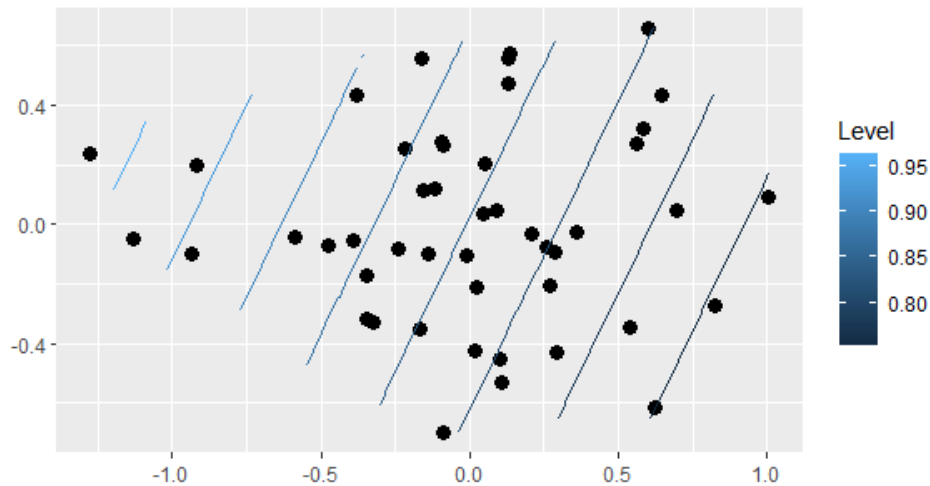
Composition varies as a function of the light environment.

Now, let's visualize how composition relates to light environment:

```
library(ggordiplots)
```

```
gg_ordisurf(ord = Oak.nmds, env.var = Oak_explan$PDIR)
```

(note that we can't customize a `gg_ordisurf()` object the same way we can with regular `ggplot2` objects)



First two axes of a three-dimensional NMDS ordination of the oak plant community. Contours connect points predicted to have the same PDIR value, with lighter colors indicating higher values.

Sample units on one end of the ordination space tend to have higher light availability than those on the other end of the ordination space. We'll use TITAN to explore the species associated with these differences.

Conclusions

This is a brief introduction to a large topic. How these techniques are used will have to reflect the details of your study.

References

Hu, Y-J., and G.A. Satten. 2020. Testing hypotheses about the microbiome using the linear decomposition model (LDM). *Bioinformatics* 36:4106-4115.

Media Attributions

- oak.nmds.Grazing
- oak.nmds.PDIR

44. SIMPER

Learning Objectives

To understand the calculations behind SIMPER.

To demonstrate how SIMPER quantifies the average contribution of each species to the difference between two groups.

To visualize community patterns in the contribution of species to compositional differences between groups.

Readings

(optional) Clarke (1993, p. 127-130)

Key Packages

```
require(vegan, tidyverse)
```

Introduction

SIMPER is intended for comparisons among levels of a categorical variable, where the response matrix is expressed as a distance matrix, particularly the Bray-Curtis dissimilarity measure.

We will illustrate it using our oak example, and comparing compositions between the three levels of our Grazing factor. See the beginning of this section for details about how to create these objects.

Bray-Curtis Dissimilarity

Recall that compositional data are often summarized using the Bray-Curtis dissimilarity measure:

$$D_{i,h} = \frac{\sum_{j=1}^p |a_{ij} - a_{hj}|}{\sum_{j=1}^p a_{ij} + \sum_{j=1}^p a_{hj}} = 1 - \frac{2 \sum_{j=1}^p \text{MIN}(a_{ij}, a_{hj})}{\sum_{j=1}^p a_{ij} + \sum_{j=1}^p a_{hj}} = 1 - \frac{2 \sum_{j=1}^p \text{MIN}(a_{ij}, a_{hj})}{a_{i\cdot} + a_{h\cdot}}$$

where

- p is the total number of species
- a_{ij} is the abundance of species j in sample unit i
- a_{hj} is the abundance of species j in sample unit h
- $a_{i\cdot}$ is the total abundance of all species in sample unit i
- $a_{h\cdot}$ is the total abundance of all species in sample unit h

This is the same formula we saw in the section about distance measures. For our purposes here, note the summations:

- In the numerator, a difference is calculated for each species and then those differences are summed together
- In the denominator, the total abundance is summed for each sample unit and then those abundances are summed together.

Clarke (1993) observed that this formula means that each species contributes uniquely to the dissimilarity between two sample units. In other words, we can consider how much of the total dissimilarity between two sample units is due to each species.

When there are multiple sample units, there are also multiple dissimilarities (i.e., pairwise combinations) to consider – each species can contribute to the dissimilarity between each pair of sample units. If our purpose is to quantify the contribution of each species to the differences between two groups, we have to consider all of these dissimilarities.

The average dissimilarity for each pairwise combination can be calculated directly via the `vegan::meandist()` function:

```
meandist(dist = vegdist(Oak1),
  grouping = Oak_explan$Grazing)
```

```
      Always      Never      Past
Always 0.6876976 0.7262910 0.6730906
Never  0.7262910 0.6872665 0.6924324
Past   0.6730906 0.6924324 0.6935371
attr(,"class")
[1] "meandist" "matrix"
attr(,"n")
grouping
Always  Never   Past
    17     24     6
```

The distances shown here are a symmetric square matrix where each value is the average distance between two sample units. Values on the diagonal are the average distances between observations in the same group; all other values are the average distances between an observation in one group and an observation in another group. For example, the largest average distance here, 0.726, is between the 'Always' and 'Never' groups.

Similarity percentage (SIMPER) partitions the Bray-Curtis dissimilarity as described above for every pair of sample units, and then calculates the average contribution of each species to the difference between the sample units. These contributions are relativized so that the average contributions of all species sum to 1. Statistical significance of these contributions is assessed by permuting the group identities.

Note that the title is misleading: a high value for SIMPER means that a species has a high contribution to the difference between the two groups, not that it has high similarity between them.

Published SIMPER Examples

Encarnação et al. (2015) compared subtidal soft-bottom macrofaunal assemblages in areas influenced or not influenced by submarine groundwater discharge.

Evangelista et al. (2016) compared vegetation data from 1972 and 2014, and used SIMPER to identify species that exhibited strong temporal changes.

da Costa et al. (2018) used SIMPER and PCA to examine plant growth promoting bacteria. In particular they calculated the SIMPER value for each treatment-control pair and used this value as an index of invasion.

Gibert & Escarguel (2019) consider ways that SIMPER patterns can indicate whether community assembly is dominated by niche- or dispersal-related processes.

SIMPER in R (`vegan::simper()`)

SIMPER is available in the `vegan` package. Its usage is:

```
simper(comm,
  group,
  permutations = 999,
  parallel = 1,
  ...
)
```

The arguments include:

- `comm` – a compositional data object. Required.
- `group` – a grouping variable. Usually included. If excluded, this function returns the average contribution of each species to the average pairwise dissimilarity in the dataset.
- `permutations` – number of permutations to perform when testing if the average contribution of that species when the group identities are randomly permuted.

The output of this function is a list (collection of dataframes). Each dataframe pertains to a different pairwise combination of groups (e.g., levelA vs. levelB) and is named following the format 'levelA_levelB'. Notes about the output in each dataframe:

- Species are in descending order – the species that contributes the most is listed first.
- Column descriptions:
 - **average** = average contribution of this species to the average dissimilarity between observations from the two groups. The sum of this column is the average dissimilarity between observations from the two groups.
 - **sd** = standard deviation of the contribution of this species (i.e., based on its contribution to all dissimilarities between observations from the two groups).
 - **ratio** = ratio of average to sd. Basically, a coefficient of variation (CV).
 - **ava**, **avb** = average abundance of this species in each of the two groups. Only included if a grouping variable (**group**) was included in the original call to **simper()**.
 - **cumsum** = cumulative contribution of this and all previous species in list. Based on **average**, but expressed as a proportion of the average dissimilarity. As a result, the maximum value of this column is 1.
 - **p** = permutation-based p-value; probability of getting a larger or equal average contribution for each species if the grouping factor was randomly permuted.

Oak Example

Applying SIMPER to our data:

```
simper.Grazing <- simper(Oak1, Oak_explan$Grazing)
```

In our example, there are three levels and thus three pairwise combinations (i.e., $n(n-1)/2$). We can extract information separately for each pairwise combination by indexing the relevant dataframe within the list. For example:

```
summary(simper.Grazing)$Always_Past %>%  
  round(3) %>%  
  head()
```

	average	sd	ratio	ava	avb	cumsum	p
Frvi	0.023	0.022	1.059	0.588	0.400	0.034	0.748
Trla	0.022	0.023	0.938	0.471	0.333	0.067	0.330
Mebu	0.020	0.022	0.905	0.376	0.333	0.096	0.285
Acmi	0.019	0.020	0.956	0.412	0.233	0.125	0.070
Rhdi.s	0.014	0.010	1.373	0.358	0.533	0.146	0.078
Nepa	0.014	0.020	0.677	0.235	0.167	0.166	0.224

For example, 'Frvi' is the species that contributes the largest amount of the difference between these two groups. The cumulative contribution of 'Frvi' (**cumsum**) is its average contribution (i.e., **average**) divided by the overall average dissimilarity between these two groups (calculated above):

$0.023 / 0.673 = 0.034$.

We can also re-organize these results into a single dataframe. While doing so here, I'll also keep track of the order of the species within each comparison.

```
comparisons <- c("Always_Past", "Always_Never", "Past_Never")

simper.results <- c()

for(i in 1:length(comparisons)) {
  require(tidyverse)
  temp <- summary(simper.Grazing)[as.character(comparisons[i])] %>%
  as.data.frame()
  colnames(temp) <- gsub(
    paste(comparisons[i], ".", sep = ""), "", colnames(temp))
  temp <- temp %>%
  mutate(Comparison = comparisons[i],
    Position = row_number()) %>%
    rownames_to_column(var = "Species")
  simper.results <- rbind(simper.results, temp)
}
```

	Species	average	sd	ratio	ava	avb	cumsum
1	Frvi	0.02285920	0.02158049	1.0592528	0.5882353	0.4000000	0.03396154
2	Trla	0.02198060	0.02342799	0.9382194	0.4705882	0.3333333	0.06661777
3	Mebu	0.02008343	0.02220031	0.9046462	0.3764706	0.3333333	0.09645539
4	Acmi	0.01903871	0.01992442	0.9555464	0.4117647	0.2333333	0.12474090
5	Rhdi.s	0.01435717	0.01045780	1.3728664	0.3584131	0.5329457	0.14607111
6	Nepa	0.01360838	0.02011243	0.6766152	0.2352941	0.1666667	0.16628886
	p	Comparison	Position				
1	0.721	Always_Past	1				
2	0.356	Always_Past	2				
3	0.283	Always_Past	3				
4	0.096	Always_Past	4				
5	0.069	Always_Past	5				
6	0.218	Always_Past	6				

Our dataset includes 103 species and there are 3 pairwise comparisons of grazing treatments, so the `simper.results` object has 309 rows.

Having the data in this format permits easy indexing and summarizing of information as illustrated below.

Using SIMPER Results

Focus on Individual Species

We can use the SIMPER results to compare responses across pairwise combinations. For example, we could focus on an individual species and compare it's importance for different pairwise comparisons:

```
simper.results %>%
  filter(Species == "Frvi")
```

	Species	average	sd	ratio	ava	avb	cumsum
1	Frvi	0.02285920	0.02158049	1.0592528	0.5882353	0.4000000	0.03396154
2	Frvi	0.02484323	0.02468804	1.0062858	0.5882353	0.3333333	0.03420561
3	Frvi	0.02423108	0.02526568	0.9590513	0.4000000	0.3333333	0.07028970

	p	Comparison	Position
1	0.721	Always_Past	1
2	0.260	Always_Never	1
3	0.511	Past_Never	2

'Frvi' is one of the species that contributes most to the differences between sample units from different groups, though it is not statistically significant (probably because of its high variability among sample units).

We can also filter the results to focus on those that are statistically significant:

```
simper.results %>%
  filter(p <= 0.05) %>%
  select(Species, average, Comparison, Position)
```

	Species	average	Comparison	Position
1	Aqfo	0.013522186	Always_Past	8
2	Trla	0.023175569	Always_Never	2
3	Syal.s	0.021775672	Always_Never	3
4	Acmi	0.017852658	Always_Never	5
5	Roeg.s	0.009986365	Always_Never	13
6	Plla	0.007682389	Always_Never	36
7	Adbi	0.007646766	Always_Never	37
8	Taof	0.007643184	Always_Never	38
9	Lope	0.007363360	Always_Never	40
10	Popr	0.006859677	Always_Never	43
11	Quga.s	0.006552076	Always_Never	48
12	Kocr	0.006200026	Always_Never	50
13	Daca	0.005903157	Always_Never	53
14	Erla	0.004826316	Always_Never	69
15	Elgl	0.004818007	Always_Never	71
16	Feru	0.004744426	Always_Never	73
17	Agte	0.004352958	Always_Never	80
18	Arme.s	0.004247614	Always_Never	82
19	Pogr	0.003994785	Always_Never	86
20	Syal.s	0.024439782	Past_Never	1
21	Rhdi.s	0.016881490	Past_Never	6
22	Aqfo	0.016242687	Past_Never	7
23	Sado	0.014907039	Past_Never	9
24	Frbr	0.009676594	Past_Never	20
25	Pyfu.s	0.009057556	Past_Never	28

Many more species have significant values in the Always_Never combination (18) than in the Always_Past combination (1) or the Past_Never combination (6) – an indication that differences are strongest between these two groups?

A few species are identified as important in multiple comparisons. For example, 'Aqfo' is significant in the 'Always_Past' and 'Past_Never' comparisons – this may be an indication that its abundance patterns are different in the 'Past' group than in the 'Always' or 'Never' groups. However, SIMPER doesn't indicate which group a species was more strongly associated with; see Indicator Species Analysis for that.

What do you think about the distribution patterns of 'Syal.s'?

Statistical significance in SIMPER is very sensitive to abundance patterns – see 'Conclusions' below.

Adding up the contributions of all species in each pairwise combination:

```
simper.results %>%  
  group_by(Comparison) %>%  
  summarize(sum.average = sum(average))
```

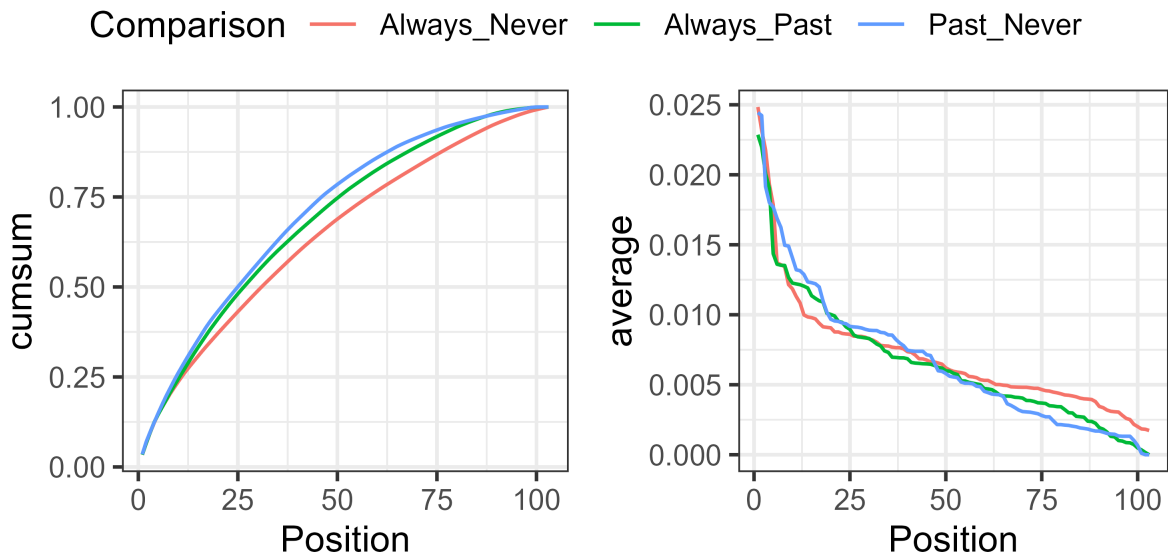
```
# A tibble: 3 × 2  
  Comparison sum.average  
  <chr>      <dbl>  
1 Always_Never      0.726  
2 Always_Past       0.673  
3 Past_Never        0.692
```

Compare to the results from `meandist()` above and verify that these totals are the same.

Examine Patterns Across Species

Another way to use these data would be to consider whether the differences are 'driven' by a subset of the species or reflect small contributions from lots of species. Gibert & Escarguel (2019) propose that this can be used to determine whether species distributions are driven by niche- or dispersal-assembly processes.

For example, the cumulative and average contributions could be graphed and compared against pairs of groups:



SIMPER comparing sample units from the oak plant community that were part of three different grazing histories. Along the horizontal axis, species (103) are arranged from highest to lowest contribution to the average compositional difference between two grazing histories. The image on the left shows the cumulative contribution of each species to the average Bray-Curtis dissimilarity between sample units in each pair of grazing histories. A straight diagonal line would indicate equal contributions by species. The image on the right shows the average contribution of each species to the average Bray-Curtis dissimilarity between sample units in each pair of grazing histories. If this value was zero, it would mean that a species did not contribute to the difference (i.e., had equal abundance in both groups).

In this case, the cumulative contribution of species is more even for the Always_Never comparison than for the others (left), and every species contributes some non-zero amount to the compositional difference (right).

Code to create the above graphic follows:

```
p1 <- ggplot(data = simper.results,
  aes(x = Position, y = cumsum)) +
  geom_line(aes(colour = Comparison)) +
  theme_bw()

p2 <- ggplot(data = simper.results,
  aes(x = Position, y = average)) +
  geom_line(aes(colour = Comparison)) +
  theme_bw()

library(ggpubr)

ggarrange(p1, p2, common.legend = TRUE)

ggsave("graphics/simper.Grazing.png", width = 5, height = 2.5,
  units = "in", dpi = 600)
```

Conclusions

The help files (`?simper`) provide some caveats for the interpretation of SIMPER. For example:

“The method gives the contribution of each species to overall dissimilarities, but these are caused by variation in species abundances, and only partly by differences among groups. Even if you make groups that are copies of each other, the method will single out species with high contribution, but these are not contributions to non-existing between-group differences but to random noise variation in species abundances.”

In general, this approach doesn't appear to have received much attention from ecologists – perhaps because of these concerns about how to interpret the results – though the work of Gibert & Escarguel (2019) suggests that other ways to use this information could be developed. For example, comparing patterns within groups to patterns between groups might be a fruitful way forward...

Each pair of groups is considered separately in SIMPER, whereas Indicator Species Analysis allows you to compare multiple groups to one another.

References

Clarke, K.R. 1993. Non-parametric multivariate analyses of changes in community structure. *Australian Journal of Ecology* 18:117-143.

da Costa, P.B., S.B. de Campos, A. Albersmeier, P. Dirksen, A.L.P. Dresseno, O.J.A.P. dos Santos, K.M.L. Milani, R.M. Etto, A.C. Battistus, A.C.P.R. da Costa, A.L.M. de Oliveira, C.W. Galvão, V.F. Guimarães, A. Sczyrba, V.F. Wendisch, and L.M.P. Passaglia. 2018. Invasion ecology applied to inoculation of plant growth promoting bacteria through a novel SIMPER-PCA approach. *Plant and Soil* 422:467-478.

Encarnação, J., F. Leitão, P. Range, D. Piló, M.A. Chícharo, and L. Chícharo. 2015. Local and temporal variations in near-shore macrobenthic communities associated with submarine groundwater discharges. *Marine Ecology* 36(4):926-941.

Evangelista, A., L. Frate, M.L. Carranza, F. Attore, G. Pelino, and A. Stanisci. 2016. Changes in composition, ecology and structure of high-mountain vegetation: a re-visitation study over 42 years. *AoB Plants* 8:plw004.

Gibert, C., and G. Escarguel. 2019. PER-SIMPER—A new tool for inferring community assembly processes from taxon occurrences. *Global Ecology and Biogeography* 28(3):374-385.

Media Attributions

- `simper.Grazing`

45. ISA

Learning Objectives

To understand the calculations behind ISA:

- Classical approach based on abundance data
- Considerations of combinations of groups
- Simplified approach based on presence/absence data

To summarize and apply the results of an ISA.

To consider additional ways that ISA could be used.

Readings (recommended)

Dufrêne & Legendre (1997)

Key Packages

```
require(vegan, indicpecies)
```

Introduction

Indicator Species Analysis (ISA) was originally developed by Dufrêne & Legendre (1997) as a technique to identify species that are strongly associated with a particular group. One motivation for developing this technique was to provide a tool for determining how many groups to focus on in a cluster analysis.

ISA has been extended in several ways since it was originally described:

- De Cáceres & Legendre (2009) showed how to use ISA to identify species that are associated with multiple groups.

- I (Bakker 2008) demonstrated that ISA produces very similar results if applied to presence/absence data rather than abundance data. Implementation is identical, but with a matrix of presence/absence data instead of abundance data.
- In the same paper, I also showed how ISAs from different sites can be combined. I will not explore this idea further here, but it is available through `indicspecies::signassoc()`.
- Baker & King (2010) developed TITAN to identify species that are associated with high or low values of a continuously distributed variable.

I will explain the way this was first proposed (i.e., the classical ISA) and then the extensions to multiple groups and to presence/absence data. I consider TITAN separately.

The Classical ISA

ISA is a remarkably simple technique that I think is underappreciated. Advantages include:

- easy to calculate
- easy to interpret
- statistical significance is assessed via permutation tests
- can be used with any categorical classification (*a priori* or *a posteriori*)
- is calculated independently for each species

ISA requires a **typology**, or classification of sample units into groups. As noted below, multiple groupings can be tested, and these groupings can be hierarchical or non-hierarchical.

ISA involves calculating the **specificity** (A_{ij} ; relative abundance) and **fidelity** (B_{ij} ; relative frequency) of species i in group j . These values are then multiplied together to yield the test statistic, the **Indicator Value** (IV_{ij}).

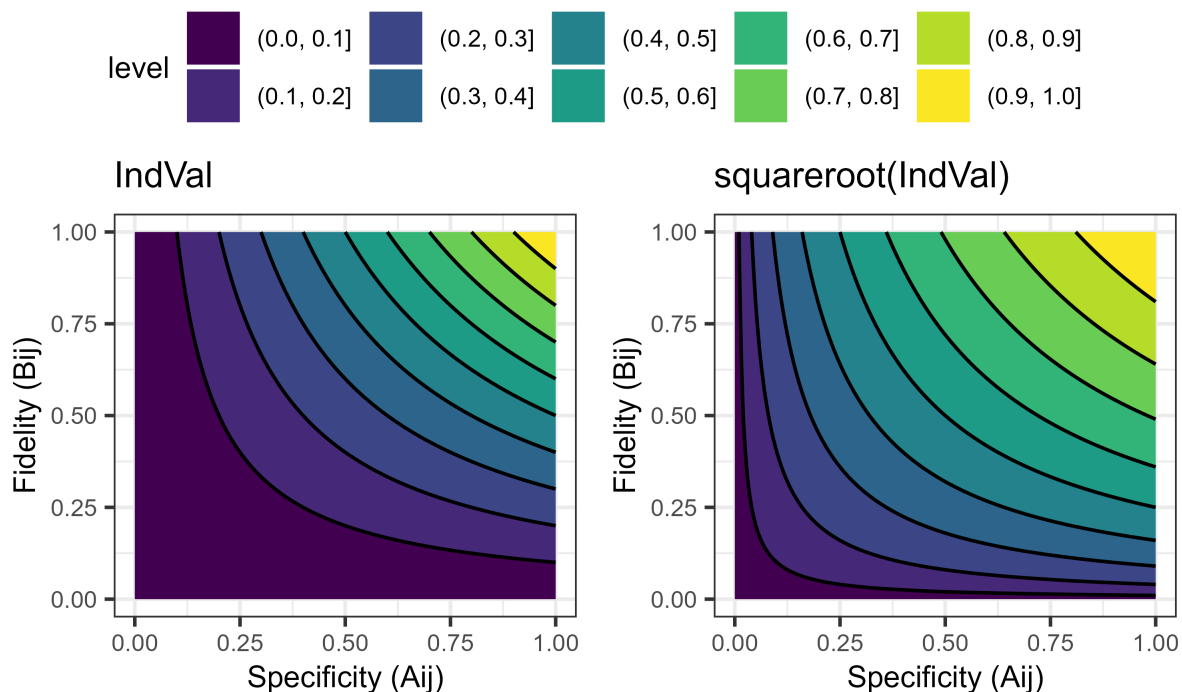
	Formula	Verbal Interpretation	Range
Specificity / Relative Abundance:	$A_{ij} = \frac{\bar{x}_{ij}}{\sum_j \bar{x}_{i\bullet}}$	Mean cover of species i in group j as a proportion of its mean cover in all groups	0 to 1
Fidelity / Relative Frequency:	$B_{ij} = \frac{n_{ij}}{n_{\bullet j}}$	Proportion of plots in group j on which species i occurs	0 to 1
Indicator Value:	$IV_{ij} = A_{ij} \times B_{ij} \times 100$	As proposed by Dufrêne & Legendre (1997)	0 to 100
	$IV_{ij} = \sqrt{A_{ij} \times B_{ij}}$	As reported in <code>indicspecies::multipatt()</code>	0 to 1

where

- \bar{x}_{ij} is the mean cover of species i within group j
- $\sum_j \bar{x}_{i\bullet}$ is the sum of the mean cover of species i in all groups
- n_{ij} is the number of plots in group j occupied by species i
- $n_{\bullet j}$ is the total number of plots in group j .

The key statistic is the Indicator Value (IndVal or IV or IV_{ij}). Some notes about its interpretation:

- IV_{ij} is 0 when species i is absent from group j , and is at its maximum (1 or 100) when species i occurs on all plots within group j and does not occur in other groups.
- The highest IV_{ij} values will occur for species that are common in one group and absent from other groups within the typology.
- Uncommon species will have low B_{ij} values and therefore low IV_{ij} values.
- Ubiquitous species (i.e., species present in all groups at a given level of the typology) have high B_{ij} values and therefore high IV_{ij} values.
- Dufrêne & Legendre (1997) suggest that species i be considered a 'strong' indicator of group j if $IV_{ij} > 25$. There are multiple ways to arrive at this value. For example:
 - If a species is present on at least 50% of the plots in group j (i.e., $A_{ij} = 0.5$) and also has at least 50% of its total abundance in group j (i.e., $B_{ij} = 0.5$)
 - If a species is present on at least 25% of the plots in group j (i.e., $A_{ij} = 0.25$) and also has all of its total abundance in group j (i.e., $B_{ij} = 1.0$)
 - If a species is present on all plots in group j (i.e., $A_{ij} = 1.0$) and also has at least 25% of its total abundance in group j (i.e., $B_{ij} = 0.25$)
- The minimum of $IV_{ij} = 25$ for a strong indicator as proposed by Dufrêne & Legendre (1997) is equivalent to $IV_{ij} = 0.5$ in the formulation reported by `indicspecies::multipatt()` which takes the square root of this value.
- The figure below illustrates isoclines of IV :
 - Left: product of A_{ij} and B_{ij} (though expressed as a proportion rather than a percentage), as proposed by Dufrêne & Legendre (1997)
 - Right: squareroot of product of A_{ij} and B_{ij} , as reported by `indicspecies::multipatt()`



Isoclines of the Indicator Value (IndVal or IV). IV is the product of Specificity and Fidelity (left) or the square root of that product (right). Its maximum value is 1, which occurs when both Specificity and Fidelity also equal 1. The code to create this figure is provided at the end of the chapter.

Species i is tentatively considered an indicator of the group j in which its IV is largest. The statistical significance of these values is assessed using permutation tests. These tests are necessary because

ubiquitous species have high IV_{ij} values but are not helpful in distinguishing one group from another.

Indicators of Combinations of Groups

As originally proposed, ISA only considers species as potential indicators of one group or another. For example, imagine that you had obtained samples from forest, grassland, and river. The classical ISA identifies species that are indicators of forest OR grassland OR river. However, a species that occurred in both forest AND grassland would not be identified as an indicator of either habitat.

Miquel De Cáceres and colleagues (De Cáceres & Legendre 2009; De Cáceres et al. 2010) extended the classical ISA to enable the identification of indicators of combinations of groups. They argued that this extension provides a way to identify species with differing niche breadths (i.e., niches spanning more than one category).

For example, in the habitat example that I just described, the method proposed by De Cáceres et al. tests for indicators of individual habitats (forest OR grassland OR river) and for indicators of combinations of habitats (e.g., (forest + grassland) OR river). The `restcomb` argument within `indicspecies::multipatt()` provides control over which combinations are considered.

Simplified ISA: Presence/Absence Data

Dufrêne & Legendre (1997) also proposed a simplified ISA based on presence/absence data. For the simplified ISA, the formula for specificity (A_{ij}) is modified to:

$$A_{ij} = \frac{n_{ij}}{n_{i\bullet}}$$

where

- n_{ij} is the number of plots in group j occupied by species i
- $n_{i\bullet}$ is the total number of plots occupied by species i

Verbally, specificity is now the proportion of plots occupied by species i that are located in group j . B_{ij} and IV_{ij} are calculated as above.

In practice, a simplified ISA can be implemented by transforming the data to binary data (i.e., 0 = absence, 1 = presence) and calculating the ISA on the transformed data. If a species is equally abundant on all plots where it occurs, the classical and simplified ISAs yield identical results. As variability in abundance among plots increases, the classical and simplified ISAs give differing weights to A_{ij} , and therefore yield different results.

A potentially large advantage of the simplified ISA is that the data are easily summarized and displayed. To summarize the data necessary for the classical ISA, you need the raw data (i.e., abundance of species i on every plot j). In contrast, the data needed for the simplified ISA can be succinctly summarized. Referring to the formulae above, you can see that only three values are required to calculate the simplified ISA for a given species:

- n_{ij} = the number of plots in group j occupied by species i

- $n_{i\bullet}$ = the total number of plots occupied by species i
- $n_{\bullet j}$ = the total number of plots in group j

The simplified ISA can be conducted using the classical approach (each group separately) or the extension to combinations of groups.

In an earlier study (Bakker 2008), I used classical (abundance-based) and simplified (presence/absence-based) ISAs to identify indicators of grazing treatment (Figure 2; reproduced below). There were 206 unique tests, each consisting of one species at one site. Summary conclusions:

- In 94% (193 of 206) of tests, the classical and simplified ISAs identified the same grazing treatment as having the maximum IV.
- In no test did the two methods identify opposite grazing treatments as statistically significant indicators.
- Overall, 62 species were identified as statistically significant indicators of grazing treatment. Of these:
 - 85% (53 of 62) were identified by both methods
 - 13% (8 of 62) were identified by the classical ISA only
 - 2% (1 of 62) were identified by the simplified ISA only

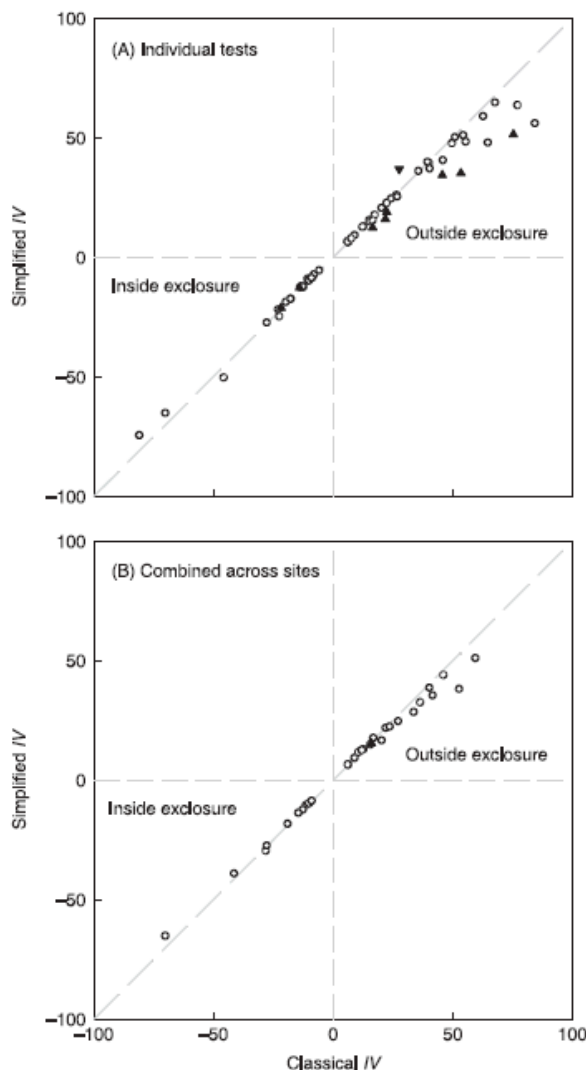


Fig. 2. Relationship between Indicator Values (IV) obtained using classical ISA and simplified ISA based on (A) tests conducted separately for each species at each site, and (B) tests combined across all sites. For simplicity, results are only shown for species identified as statistically significant indicators by the classical ISA (▲), simplified ISA (▼), or both methods (○). Indicators of conditions inside enclosures have been expressed as negative values to distinguish them from indicators of conditions outside grazing enclosures. All indicator species are listed in Supporting Information Table S2. Dashed horizontal, vertical, and 1:1 lines are shown for reference.

Published ISA Examples

Zald et al. (2020) identified plant species associated with different seasons and intervals of burning.

Pillsbury et al. (2019) used ISA to identify diatoms associated with high and low nitrogen and phosphorus conditions.

Martin & Hamman (2016) used ISA to identify the fire severity class associated with each of several plant life forms.

Severns & Sykes (2020) propose that phytopathologists use ISA to identify the fungi associated with plant diseases.

ISA in R (`indicspecies::multipatt()`)

There are several functions that can conduct ISA in R. They vary in some details, such as how IV is reported. We will focus on `indicspecies::multipatt()` but you are welcome to explore others such as `labdsv::duleg()` and `adiv::dbMANOVAspecies()` (Ricotta et al. 2021) if interested.

The usage of `multipatt()` is:

```
multipatt(x,
  cluster,
  func = "IndVal.g",
  duleg = FALSE,
  restcomb = NULL,
  min.order = 1,
  max.order = NULL,
  control = how(),
  permutations = NULL,
  print.perm = FALSE
)
```

The key arguments are:

- `x` – the plot x species data table
- `cluster` – a vector consisting of the group to which each plot belongs. Although this is called ‘cluster’, the groups do not have to come from a cluster analysis.
- `func` – function to calculate the strength of the association of each species with the groups. Options are "IndVal", "IndVal.g", "r", and "r.g". IndVal and IndVal.g are the IV mentioned above, while r and r.g are correlations. The ‘g’ suffix indicates that they include a correction for sample sizes, and are the recommended functions to use. IndVal.g is the default.
- `duleg` – whether to consider groups of sites (`FALSE`; default) or not (`TRUE`). Argument name is an abbreviation of Dufrêne & Legendre (1997), who only considered indicators of individual sites. If `duleg = TRUE`, this conducts the classical ISA that they originally proposed.
- `restcomb` – a vector of integers that restricts attention to groups of sites that make ecological sense to the analyst. Not applicable if doing a classical ISA.
- `min.order` – minimum number of sites (i.e., levels of a factor) to include in groups of sites. Default is 1 (i.e., individual sites can be their own groups).
- `max.order` – maximum number of sites (i.e., levels of a factor) to include in groups of sites. For example, if `max.order = 2`, then two levels can be combined into a group but a group including three levels will not be tested.
- `control` – options for controlling how permutations are conducted. This uses the same

`permute::how()` function that we saw in our group comparison methods.

The resulting object is of class 'multipatt' with several elements, including:

- `comb` – a plot x combination matrix, with a column for every combination of groups tested. The values within this matrix are 1 if that plot is included in that combination of groups and 0 otherwise. The column number is the integer value by which these combinations are referenced in other elements.
- `A` – a species x combination matrix, with a column for every combination of groups tested. The values are the specificity component of IV for that species in that combination of groups.
- `B` – a species x combination matrix, with a column for every combination of groups tested. The values are the fidelity component of IV for that species in that combination of groups
- `sign` – a data table summarizing the results for each species:
 - The 'index' column contains an integer denoting the combination of groups in which the species had its maximum IV.
 - The 'stat' column reports the square root of the IV.
 - The 'p.value' column reports the permutation-based p-value associated with `stat`. Note that there is no p-value when the maximum IV is in the largest combination of groups (i.e., all plots in a single group) as there are no permutations possible.

Applying `summary()` to the resulting object focuses on those species that meet desired criteria. The default is to show those species that are statistically significant at $\alpha = 0.05$. See `?summary.multipatt` for details of other criteria.

One caution about only viewing statistically significant taxa is that the p-value is permutation-based and therefore can vary; so species may be identified as significant indicators in one run but not in another run. Options to minimize this issue are to use a set group of permutations (`set.seed()`) and/or to conduct a large number of permutations.

Oak Example: Indicators of Grazing Treatments

Let's see which species are strongly associated with one grazing status over the others. We will do this using the three approaches described above: classical ISA, ISA with multiple groups, and simplified ISA (also with multiple groups).

To ensure that results are directly comparable amongst analyses, we will use the same set of permutations in each analysis.

Classical ISA

To use the classical ISA, we specify `x` and `cluster`, and include the argument '`duleg = TRUE`':

```
library(indicspecies)

set.seed(42)

Graz.ISA <- multipatt(x = Oak1,
```

```
cluster = Oak_explan$Grazing,
duleg = TRUE)
```

```
summary(Graz.ISA)
```

```
Multilevel pattern analysis
-----

Association function: IndVal.g
Significance level (alpha): 0.05

Total number of species: 103
Selected number of species: 6
Number of species associated to 1 group: 6
Number of species associated to 2 groups: 0

List of species associated to each combination:

Group Always #sps. 2
      stat p.value
Popr 0.74      0.04 *
Agte 0.72      0.03 *

Group Never #sps. 2
      stat p.value
Prav.s 0.834    0.005 **
Syal.s 0.757    0.005 **

Group Past #sps. 2
      stat p.value
Rhdi.s 0.690    0.015 *
Aqfo  0.544     0.035 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Six of the 103 species are identified as significant indicators. All such indicators are associated with a single group (as designed in the classical ISA). In this case:

- Popr and Agte are indicators of always grazed stands
- Prav.s and Syal.s are indicators of never grazed stands
- Rhdi.s and Aqfo are indicators of stands that were grazed only in the past

Note that it is simply chance that there are equal numbers of indicators for each grazing treatment. There could easily be uneven numbers of indicators, and groups for which no indicators are identified.

Each species is identified by the unique code that named its column. We can use these codes to look their names in our `Oak_spp` object and then draw on our ecological knowledge of the system to interpret them.

ISA with Multiple Groups

Our grazing example includes three levels that can be combined into seven groups:

Group	Possible Ecological Interpretation
Always	Continually grazed
Never	Never grazed
Past	No longer grazed
Never+Past	Not currently grazed
Always+Never	No change in grazing practice?
Always+Past	Grazed in the past, regardless of current grazing practices
Always+Never+Past	All plots

(these combinations of groups are identified in `Graz.ISA.2$comb` below)

The `multipatt()` function automatically combines the levels of our grouping variable into groups of levels. If desired, we can use the `restcomb` argument to restrict our attention to particular combinations of groups. For example, we might decide that the group 'Always+Never' is ecologically questionable and therefore exclude it from consideration (I have not done so here, however).

ISA with multiple groups is the default in `multipatt()`:

```
set.seed(42)
```

```
Graz.ISA.2 <- multipatt(x = Oak1,
  cluster = Oak_explan$Grazing)
```

```
summary(Graz.ISA.2)
```

```
Multilevel pattern analysis
```

```
-----
```

```
Association function: IndVal.g
```

```
Significance level (alpha): 0.05
```

```
Total number of species: 103
```

```
Selected number of species: 9
```

```
Number of species associated to 1 group: 1
```

```
Number of species associated to 2 groups: 8
```

```
List of species associated to each combination:
```

```
Group Past #sps. 1
```

```
stat p.value
```

```
Aqfo 0.544 0.035 *
```

```
Group Always+Past #sps. 6
```

```
stat p.value
```

```
Sado 0.841 0.005 **
```

```
Popr 0.808 0.010 **
```

```
Elgl 0.786 0.030 *
```

```
Hola 0.753 0.035 *
```

```
Agte 0.727 0.045 *
```

```
Acmi 0.606 0.015 *
```

```
Group Never+Past #sps. 2
```

```

      stat p.value
Prav.s 0.873    0.015 *
Ronu.s 0.705    0.045 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

In this analysis, 9 of the 103 species are identified as indicators. Some indicators are associated with 1 group and some are associated with 2 groups. In this case there are:

- One indicator (Agfo) of stands that were only grazed in the past
- Six indicators (Sado, Popr, Elgl, Hola, Agte, and Acmi) of stands that were either always grazed or only grazed in the past (i.e., not never grazed)
- Two indicators (Prav.s, Ronu.s) of stands that were not currently grazed (i.e., either Never or Past grazed)

Allowing for indicators to be identified across multiple groups has changed some of the indicators. For example, there are no indicators of stands that were only Never grazed whereas in the above analysis two species were indicators of this group. Aqfo is still identified as an indicator of a single group (Past), so its test statistic is identical here to what was calculated with the Classical ISA.

Simplified ISA (Presence/Absence Data)

To analyze these data using presence/absence data, we need to relativize the data in that way first. We'll use the multi-group ISA here.

```

Oak1.pres <- decostand(Oak1, "pa")

set.seed(42)

Graz.ISA.pres <- multipatt(x = Oak1.pres,
  cluster = Oak_explan$Grazing)

summary(Graz.ISA.pres)

```

```

Multilevel pattern analysis
-----

Association function: IndVal.g
Significance level (alpha): 0.05

Total number of species: 103
Selected number of species: 8
Number of species associated to 1 group: 2
Number of species associated to 2 groups: 6

List of species associated to each combination:

Group Past #sps. 2
      stat p.value
Frla.s 0.565    0.050 *

```

```

Aqfo    0.544    0.035 *

  Group Always+Past  #sps.  6
    stat p.value
Sado 0.862    0.005 **
Brla 0.844    0.005 **
Elgl 0.788    0.020 *
Popr 0.771    0.005 **
Acmi 0.609    0.010 **
Cyec 0.511    0.030 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Eight of the 103 species are identified as indicators, and some indicators are associated with 1 group and some are associated with 2 groups. In this case there are:

- Two indicators (Frla.s, Agfo) of stands that were only grazed in the past
- Six indicators (Sado, Brla, Elgl, Popr, Acmi, and Cyec) of stands that were either always grazed or only grazed in the past (i.e., not never grazed)

Results based on presence/absence data are broadly similar to, but not identical, to those above where we also incorporated species abundance information.

Summarizing ISA Results

An interesting element is to compare analyses and identify which indicators have changed in status. This can be done on the basis of the magnitude and significance of IV. Here, we'll just focus on those species that were identified as significant indicators and compare these across the three ISAs that we conducted. I've sorted this list alphabetically by species code.

Species	Classical ISA	Multiple Group ISA	Simplified ISA
Acmi		Always+Past	Always+Past
Agte	Always	Always+Past	
Aqfo	Past	Past	Past
Brla			Always+Past
Cyec			Always+Past
Elgl		Always+Past	Always+Past
Frla.s			Past
Hola		Always+Past	
Popr	Always	Always+Past	Always+Past
Prav.s	Never	Never+Past	
Rhdi.s	Past		
Ronu.s		Never+Past	
Sado		Always+Past	Always+Past
Syal.s	Never		

Some observations:

- Aqfo was a consistent indicator of the same group (Past) regardless of how the analysis was conducted.
- Several species that were identified as indicators of a single group in the classical ISA were determined to be even stronger indicators of a combination of groups in the multiple group and/or simplified ISAs.
- Several species were not identified as indicators of a single group in the classical ISA but were identified as strong indicators of a combination of groups in the other analyses.
- Species that were identified as significant indicators in the multiple group ISA but not in the simplified ISA differ more strongly in abundance among groups.
- None of the shrubs (species codes ending in '.s') were indicators of stands that were currently grazed (Always), perhaps reflecting their sensitivity to livestock.
- Indicators of the 'Never' and 'Always' groups were only detected in the Classical ISA; most of these species were more strongly associated with a combination of groups in the other analyses.
- The complete results for each analysis includes all species regardless of their statistical significance – we could look at this to see which combination they were most strongly associated with.

Additional Applications of ISA

Hierarchical Analyses

Dufrêne & Legendre's (1997) original description of ISA was as a method to identify how many groups to focus on in a hierarchical cluster analysis. For example, they identified the number of groups in the cluster analysis at which each species achieved its largest IV, and then tallied those up – the height in the dendrogram that produced the greatest number of largest IVs was adopted as the solution and those groups were chosen.

The hierarchical structure of a dendrogram can be easily incorporated into the multi-group ISA (`restcomb` argument), combining groups in the order that they are fused in the dendrogram, and omitting those that are not supported by the dendrogram.

Non-Hierarchical Analyses

Not all data have a hierarchical typology. For example, if you were interested in two explanatory variables, you could conduct an ISA for each of them and then combine the results.

In one project, I was interested in indicators of overstory condition (either in the open or under tree canopy) and of grazing treatment (inside or outside livestock exclosures). I did an ISA of each of these, and found that some species were indicators of overstory condition, some of grazing treatment, some of a combination of the two, and some of neither:

Cross-classification of species based on indicator status based on grazing treatment (rows) and overstory condition (columns). Unpublished data, J.D. Bakker.			
	In Open	Neither	Under Canopy
Outside	<i>Achillea millefolium</i>	<i>Elymus elymoides</i>	<i>Trifolium longipes</i>
Neither	<i>Sporobolus interruptus</i>		<i>Bromus tectorum</i>
Inside	<i>Hesperostipa comata</i>	<i>Muhlenbergia montana</i>	<i>Festuca arizonica</i>

These comparisons are more ecologically nuanced than just thinking about a single factor. For example, it reflects the fact that different factors shape the distributions of different species.

Consistency of Indicators Among Sites

One aspect of ISA that hasn't received much attention to date is the consistency of indicators across replicate sites or studies. Meta-analytical techniques can be used to combine the ISA results from multiple sites and thereby assess the consistency of indicators (see Bakker 2008 for details).

For example, areas inside and outside a grazing exclosure at each of five sites in northern Arizona were sampled in 1941. I used these data to examine whether *Muhlenbergia montana*, a common grass, was an indicator of conditions inside or outside livestock exclosures. To do so, I analyzed each site separately and then combined the results into an overall value for each grazing treatment:

Site	Inside		Outside	
	IV	P-value	IV	P-value
Big Fill	36.1	0.5882	38.3	0.4060
Black Springs	46.0	0.0004	3.4	1.0000
Fry Park	92.1	0.0001	0.0	1.0000
Reese Tank	59.2	0.0563	40.8	0.9450
Rogers Lake	53.1	0.0001	0.5	1.0000
Overall	59.2	0.0001	15.5	1.0000

Although *M. montana* was clearly not a statistically significant indicator of conditions inside exclosures at one site, and was marginally significant at a second site, it was significant at three other

sites. When considered together, it was clearly an indicator of conditions inside exclosures overall. I did similar analyses with a number of other species and found that species that occurred on multiple sites were more likely to be indicators than those that occurred on a single site.

Meta-Analyses

Since ISA is calculated separately for each species, the data can also be summarized separately for each species – for example, there is no need to keep track of which plot each species occurred in. This is particularly useful for the simplified ISA, which only requires three values as noted above.

If the relevant data are reported in the literature, they can be used to assess the generality of indicator species using independent datasets. Furthermore, since ISAs are conducted independently for each species, studies can be drawn from across the distributional range of each species (Bakker 2008).

Species Combinations

De Cáceres et al. (2012) proposed that ISA could also be used to identify combinations of species that distinguish sites. This might occur, for example, if two species are functionally equivalent or even if a taxon was identified to the species level in some plots but to the genus level in other plots (e.g., *Poa pratensis*, *Poa* spp.).

This approach can be implemented using `indicspecies::combinespecies()` to create combinations of species, up to some `max.order`. See De Cáceres (2023) for details.

Recommendations for ISA

Some recommendations for using ISA, particularly with respect to the potential to combine the results of multiple sites or studies:

1. When collecting data, identify individuals to the finest taxonomic level possible. Congeneric species, for example, may differ in indicator status (Weiss & Rice 2005; Nahmani et al. 2006) and may differ in nativity and other characteristics.
2. Consider the proper exchangeable units for the permutations tests (Anderson & ter Braak 2003). Where appropriate, such as when a factor is nested within sites, sites can be analyzed separately and the results combined using meta-analytic techniques (Bakker 2008).
3. Comparisons of ISAs from multiple studies require that the same typology be used in each study. Simple pairwise comparisons (e.g., grazed vs. ungrazed) are more likely to be comparable between studies than comparisons among multiple levels. Groups included in the typology should be described in detail.
4. Ensure that the sample size within each group is sufficient to sample the vegetation. Doing so will also provide enough samples to permit an adequate number of permutations for assessing statistical significance.
5. At a minimum, report the sample size and frequency of all species in all groups, so that the simplified *IV* can be calculated. Digital appendices and other repositories permit the archiving of these data in an accessible manner (Parr & Cummings 2005).
6. To prevent publication bias in future meta-analyses (Gurevitch & Hedges 2001), publish data for all species, regardless of whether or not they were significant indicators.

What about TWINSpan?

Prior to the development of ISA, the main statistical technique for identifying species associated with groups was Two-Way INDicator SPecies ANALysis (TWINSpan) (Hill et al. 1975; McCune & Grace 2002, ch. 12). TWINSpan is a divisive approach – it starts from the assumption that everything is in one group and then looks for ways to divide that group. However, several aspects of TWINSpan greatly limit its utility:

- cannot be used for predefined groups
- performs poorly with > 1 gradient
- involves complex analytical details
- is based on Correspondence Analysis (CA) and therefore on chi-square distances
- requires “pseudospecies” to convert quantitative abundance data into qualitative presence/absence data (place abundances into classes and consider each class equivalent to a pseudospecies)

I have not seen TWINSpan used in recent years.

Conclusions

ISA is an extremely powerful technique. It has high applicability for identifying species that differ between groups.

ISA differs from SIMPER in several ways. First, ISA provides more guidance about species that differ: if a species is an indicator of one group over the other we know that it is more abundant and frequent in the first group. In contrast, SIMPER quantifies how much a species contributes to the difference between groups but doesn't indicate where the species occurs. Second, all sample units are considered simultaneously in ISA whereas the contribution of each species to each pairwise distance is calculated in SIMPER and then those contributions are averaged. This means that the connection to the actual dissimilarity between sample units is not as strong for ISA, though I have not seen explicit comparisons of ISA and SIMPER.

ISA was not designed to relate species to continuously distributed explanatory variables, though it has been adapted for that purpose as part of a technique known as TITAN.

Data adjustments remain important; these calculations will be affected by adjustments. You should use the same adjustments as in all other aspects of your analysis.

References

- Anderson, M.J., and C.J.F. ter Braak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.
- Baker, M.E., and R.S. King. 2010. A new method for detecting and interpreting biodiversity and ecological community thresholds. *Methods in Ecology and Evolution* 1:25-37.
- Bakker, J.D. 2008. Increasing the utility of Indicator Species Analysis. *Journal of Applied Ecology* 45:1829-1835.
- De Cáceres, M., and P. Legendre. 2009. Associations between species and groups of sites: indices and statistical inference. *Ecology* 90:3566-3574.

- De Cáceres, M., P. Legendre, and M. Moretti. 2010. Improving indicator species analysis by combining groups of sites. *Oikos* 119:1674-1684.
- De Cáceres, M., P. Legendre, S.K. Wiser, and L. Brotons. 2012. Using species combinations in indicator value analyses. *Methods in Ecology and Evolution* 3:973-982.
- De Cáceres, M. 2023. Indicator Species Analysis using 'indicspecies'. <https://cran.r-project.org/web/packages/indicspecies/vignettes/IndicatorSpeciesAnalysis.html>
- Dufrêne, M., and P. Legendre. 1997. Species assemblages and indicator species: the need for a flexible asymmetrical approach. *Ecological Monographs* 67:345-366.
- Gurevitch, J., and L.V. Hedges. 2001. Meta-analysis: combining the results of independent experiments. Pages 347-369 in S.M. Scheiner and J. Gurevitch (editors). *Design and analysis of ecological experiments*. Oxford University Press, New York, NY.
- Hill, M.O., R.G.H. Bunce, and M.W. Shaw. 1975. Indicator Species Analysis, a divisive polythetic method of classification, and its application to a survey of native pinewoods in Scotland. *Journal of Ecology* 63(2):597-613.
- Martin, R.A., and S.T. Hamman. 2016. Ignition patterns influence fire severity and plant communities in Pacific Northwest, USA, prairies. *Fire Ecology* 12:88-102.
- McCune, B., and J.B. Grace. 2002. *Analysis of ecological communities*. MjM Software Design, Gleneden Beach, OR.
- Nahmani, J., P. Lavelle, and J.-P. Rossi. 2006. Does changing the taxonomic resolution alter the value of soil macroinvertebrates as bioindicators of metal pollution? *Soil Biology and Biochemistry* 38:385-396.
- Parr, C.S., and M.P. Cummings. 2005. Data sharing in ecology and evolution. *Trends in Ecology and Evolution* 20:362-363.
- Pillsbury, R.W., R.J. Stevenson, M. Munn, and I.R. Waite. 2019. Relationships between diatom metrics based on species nutrient traits and agricultural land use. *Environmental Monitoring and Assessment* 191:228. <https://doi.org/10.1007/s10661-019-7357-8>
- Ricotta, C., S. Pavoine, B.E.L. Cerabolini, and V.D. Pillar. 2021. A new method for indicator species analysis in the framework of multivariate analysis of variance. *Journal of Vegetation Science* 32(2):e13013.
- Severns, P.M., and E.M. Sykes. 2020. Indicator Species Analysis: a useful tool for plant disease studies. *Phytopathology* 110(12):1860-1862.
- Smucker, N.J., E.M. Pilgrim, C.T. Nietch, J.A. Darling, and B.R. Johnson. 2020. DNA metabarcoding effectively quantifies diatom responses to nutrients in streams. *Ecological Applications* 30:e02205.
- Weiss, J.M., and S.R. Reice. 2005. The aggregation of impacts: using species-specific effects to infer community-level disturbances. *Ecological Applications* 15:599-617.
- Zald, H.S.J., B.K. Kerns, and M.A. Day. 2020. Limited effects of long-term repeated season and interval of prescribed burning on understory vegetation compositional trajectories and indicator species in ponderosa pine forests of northeastern Oregon, USA. *Forests* 11:834. <https://doi.org/10.3390/f11080834>

PS: Graphs of Indicator Value Contours

Here's the code to create the above figure showing contours for Indicator Value based on different values for A_{ij} and B_{ij} .

```

temp <- expand.grid(Aij = seq(0, 1, 0.01),
  Bij = seq(0, 1, 0.01))

temp <- temp %>%
  mutate(IVij = Aij * Bij,
    sqrt.IVij = sqrt(IVij))

IV.graph <- ggplot(data = temp,
  aes(x = Aij, y = Bij, z = IVij)) +
  geom_contour_filled() +
  geom_contour(colour = "black") +
  labs(title = "IndVal",
    x = "Specificity (Aij)",
    y = "Fidelity (Bij)") +
  theme_bw() +
  theme(text = element_text(size = 9))

sqrt.IV.graph <- ggplot(data = temp,
  aes(x = Aij, y = Bij, z = sqrt.IVij)) +
  geom_contour_filled() +
  geom_contour(colour = "black") +
  labs(title = "squareroot(IndVal)",
    x = "Specificity (Aij)",
    y = "Fidelity (Bij)") +
  theme_bw() +
  theme(text = element_text(size = 9))

ggarrange(IV.graph, sqrt.IV.graph, common.legend = TRUE)

ggsave("graphics/IndVal.contours.png",
  width = 5, height = 3, units = "in", dpi = 600)

```

Media Attributions

- IndVal.contours
- Bakker.2008_Figure2

46. TITAN

Learning Objectives

To understand how TITAN functions as a combination of a URT and ISA to relate compositional data to a continuously distributed explanatory variable.

To demonstrate how to conduct TITAN and explore the results numerically and graphically.

Readings (recommended)

Baker & King (2010)

Key Packages

```
require(tidyverse, TITAN2)
```

Introduction

SIMPER and Indicator Species Analysis can compare species patterns among categorical groups. However, what if you want to relate species abundances to continuously distributed variables such as light availability or elevation or heat load? That's where TITAN (Threshold Indicator Taxa Analysis) comes in.

TITAN

TITAN (Threshold Indicator Taxa Analysis) was proposed by Baker & King (2010, 2013; King & Baker 2010). It is a blend of univariate regression trees and ISA:

- Like a regression tree, TITAN considers all possible values that split the environmental variable into two groups:
 - The set of low values (**z-**). The species in this group respond negatively to increases in the environmental gradient, and therefore are also called **decreasers**.
 - The set of high values (**z+**). The species in this group respond positively to increases in the environmental gradient, and therefore are also called **increasers**.
- For each split of the environmental variable, TITAN conducts an ISA. This summarizes, for each species, whether it is more strongly associated with z- or z+ and the value of the Indicator Value (IndVal).

After testing all possible values, each species is assigned to the group in which it produced the highest IndVal. The value of the environmental variable that produced this separation is called the **change point**. In other words, this is the value of the environmental gradient that most strongly separated this taxon into two groups such that it was as strong as possible of an indicator for one of the two groups.

If we focused only on the observed data, we would not be able to assess statistical significance. To do so, TITAN uses bootstrap resampling. This is a permutation-based procedure that we referenced a few weeks ago in the context of random forests. If the abundance of a species is not related to the environmental variable, this could be evident in two ways:

- Species would be assigned to the z- group in some permutations and to the z+ group in other permutations.
- The species would not be a strong indicator of the group to which it was assigned.

Two metrics are used to evaluate the strength of the relationship as determined through the bootstrap resampling:

- **Purity**: is the species response consistently assigned in the same direction? In other words, is the species consistently assigned to either the z- or z+ group?
- **Reliability**: is the species a strong indicator of the group to which it is assigned? In other words, in how many of the bootstrap permutations is IndVal determined to be statistically significant?

TITAN's computations take more time than any of the other techniques that we've covered, because each species is tested individually against each potential split within the continuous variable, with bootstrapping to estimate uncertainty.

Finally, TITAN allows for the evaluation of individual species and of the community as a whole.

Published TITAN Examples

At UW's Pack Forest, McBurney et al. (2017) examined the ectomycorrhizal fungal (EMF) community of a red alder stand and used TITAN to relate the abundance of these species to several environmental variables. For example, some EMF taxa were more common at low soil moisture values and others were more common at high soil moisture values.

Smucker et al. (2020) showed that diatom OTUs relate to nutrient concentrations in streams, while Beauchene et al. (2014) related summer stream temperatures to the abundances of various fish taxa.

Martin & Hamman (2016) used TITAN to relate plant life forms to fire severity.

Morissette et al. (2019) identified a suite of bird species that responded positively to conversion of boreal forest to agriculture and a separate suite of birds that responded negatively to this conversion. Similarly, Farwell et al. (2020) identified threshold amounts of forest loss affecting songbirds.

Costas et al. (2018) related the abundances of macroinvertebrate taxa to heavy metal levels in sediments.

TITAN in R (**TITAN2::titan()**)

TITAN is available in R through the TITAN2 package. Baker et al. (2020) provide a useful tutorial to the package; I encourage you to work through their examples.

The `titan()` function is the primary wrapper function in this package. Its usage is:

```
titan(env,
      txa,
      minSplt = 5,
      numPerm = 250,
      boot = TRUE,
      nBoot = 500,
      imax = FALSE,
      ivTot = FALSE,
      pur.cut = 0.95,
      rel.cut = 0.95,
      ncpus = 1,
      memory = FALSE,
      messaging = TRUE
    )
```

The key arguments (noting that some are abbreviations that can be easily misspelled) are:

- `env` – the environmental gradient to be tested (i.e., a vector)
- `txa` – the compositional matrix
- `minSplt` – the minimum split size to use during partitioning. Default is 5. In other words, the smallest number of samples in z- or z+ will be 5; all other samples will be in the other group.
- `numPerm` – number of random permutations to be used during ISA – these are used to derive a distribution of IndVal values for each species. Default is 250, but package authors recommend increasing this to 500 or 1000 during formal analyses so there is less variation among runs. However, increasing this necessarily increases computation time.
- `boot` – whether to conduct bootstrap resampling (**TRUE**) or not (**FALSE**). Default is to do so (and is strongly recommended by package authors).
- `nBoot` – number of replicates to use during bootstrap resampling. Default is 500, but package authors recommend using 500 or 1000 for formal analyses.
- `imax` – whether to determine taxon-specific change points based on indicator value (IndVal) maxima (**TRUE**) or z-score maxima (**FALSE**; the default).
- `ivTot` – whether to calculate indicator value (IndVal) scores using total relative abundance (**TRUE**) or mean relative abundance (**FALSE**; the default).
- `pur.cut` – cut-off proportion for determining **purity** (pure response direction) across bootstrap replicates. The default (0.95) means that 95% of the results from bootstrap replicates agree with the observed response direction.
- `rel.cut` – cut-off proportion for determining **reliability** (reliable response magnitude) across bootstrap replicates. The default (0.95) means that 95% of the results from bootstrap replicates have an IndVal p-value < 0.05.

The resulting object is a list with several elements, including:

- `sppmax` – a table showing the summary statistics for each species:
 - `ienv.cp` – change point if based on `IndVal` maximum. Ignore unless you set `imax = TRUE`.
 - `zenv.cp` – change point if based on z-score maximum (default setting)
 - `freq` – frequency; number of non-zero abundance values
 - `maxgrp` – in which group the species had its maximum `IndVal`. Either 1 (z- group) or 2 (z+ group).
 - `IndVal` – the maximum `IndVal` statistic (range = 0-100%) for the species
 - `obsiv.prob` – Probability of an equal or larger `IndVal` from random permutations. Used for calculation of reliability.
 - `zscore` – `IndVal` expressed as a z-score
 - 5%, 10%, 50%, 90%, 95% – change point quantiles among bootstrap replicates
 - `purity` – proportion of replicates for which `maxgrp` matches observed `maxgrp`
 - `reliability` – proportion of replicates with `obsiv.prob` ≤ 0.05
 - `z.median` – median z-score across replicates
 - `filter` – 0 if species did not meet purity and reliability criteria, otherwise 1 (z- group) or 2 (z+ group)
- `sumz.cp` – observed community-level (i.e., across species) change point (`cp`) and quantiles associated with its distribution as determined by resampling. Values are summarized for:
 - All species associated with smaller values of the environmental variable (row 'sumz-')
 - All species associated with higher values of the environmental variable (row 'sumz+')
 - Filtered values, using only those species that were pure and reliable indicators of the group defined by smaller values of the environmental variable (row 'fsumz-').
 - Filtered values, using only those species that were pure and reliable indicators of the group defined by higher values of the environmental variable (row 'fsumz+').

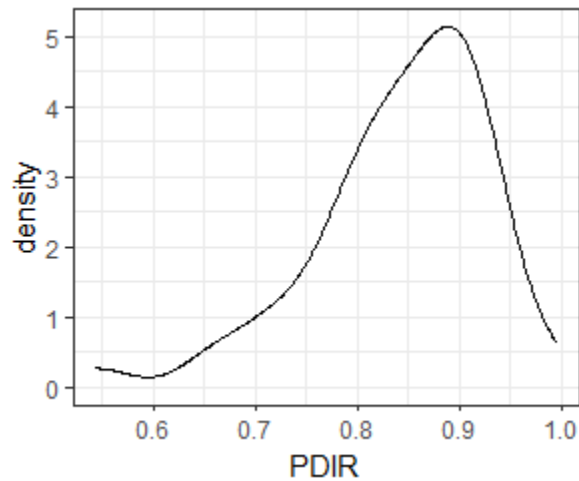
If `cp` for either filtered group differed strongly from the corresponding `cp` based on all species (e.g., 'fsumz-' vs. 'sumz-'), it would be an indication that impure or unreliable indicators were substantially affecting the scores.

Some examples of how these output can be explored tabularly and graphically are shown below.

Oak Example: Responses to PDIR

Let's relate our oak compositional data to PDIR, the potential direct incident radiation ($\text{MJ}/\text{cm}^2/\text{year}$). The stands span a relatively wide range of PDIR values:

```
ggplot(data = Oak_explan, aes(x = PDIR)) +
  geom_density() +
  theme_bw()
```

Distribution of PDIR values for the stands in the oak plant community dataset.

Furthermore, we established in the beginning of this section that PDIR is significantly associated with differences in species composition:

```
set.seed(42)
```

```
adonis2(Oak1 ~ PDIR,
  data = Oak_explan,
  distance = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ PDIR, data = Oak_explan, distance = "bray")
      Df SumOfSqs      R2      F Pr(>F)
PDIR    1   0.8068 0.06958 3.3655  0.002 **
Residual 45  10.7881 0.93042
Total   46  11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

How do species relate to this environmental gradient?

```
set.seed(42)
```

```
titan.PDIR <- titan(env = Oak_explan$PDIR,
  txa = Oak1)
```

Notes:

- We've accepted most defaults here
- The response and explanatory variable are reversed here from most (all?) other functions that we've seen
- This function will take a few minutes to run!

Individual Species Responses to PDIR

We'll start by focusing on individual species responses. We'll limit our attention to the key columns, and only display the first few species here:

```
titan.PDIR$sppmax %>%
  as.data.frame() %>%
  select(zenv.cp, freq, maxgrp, IndVal, purity, reliability, filter) %>%
  head()
```

	zenv.cp	freq	maxgrp	IndVal	purity	reliability	filter
Quga.t	0.7256032	47	1	54.19	0.534	0.660	0
Rhdi.s	0.8115082	47	2	59.67	0.624	0.812	0
Syal.s	0.9198630	46	1	93.51	0.988	0.980	1
Amal.s	0.9198630	41	1	81.56	0.466	0.910	0
GAL	0.9040936	41	1	59.70	0.580	0.578	0
Povu	0.9296631	41	1	57.53	0.542	0.684	0

For example, the light level that most strongly differentiates plots with and without Quga.t is zenv.cp = 0.7256. Quga.t is more strongly associated with PDIR values below this value (maxgrp = 1). In this group, it has an IV of 54.19. However, it is not a particularly strong indicator of this condition as bootstrap samples did not consistently identify it as an indicator of the low light group (purity = 0.534) and it was only a significant indicator in two-thirds of bootstrap samples (reliability = 0.660).

The species that respond strongly and consistently (i.e., that meet the purity and reliability criteria) are flagged with values of 1 or 2 in the 'filter' column. We can filter the results to just these species:

```
titan.PDIR$sppmax %>%
  as.data.frame() %>%
  select(zenv.cp, freq, maxgrp, IndVal, purity, reliability, filter) %>%
  filter(filter != 0) %>%
  arrange(maxgrp, zenv.cp)
```

	zenv.cp	freq	maxgrp	IndVal	purity	reliability	filter
Hodi.s	0.7256032	15	1	89.98	1.000	1.000	1
Rupa.s	0.8162860	8	1	35.22	1.000	0.954	1
Liap	0.8181559	20	1	61.53	0.992	0.960	1
Rhpu.t	0.8328414	6	1	33.33	0.996	0.968	1
Osce.s	0.8737474	23	1	59.07	1.000	0.978	1
Pomu	0.8785627	38	1	92.87	1.000	1.000	1
Cipa	0.8785627	10	1	35.71	0.998	0.960	1
Tegr	0.8972871	24	1	65.22	1.000	0.998	1
Syal.s	0.9198630	46	1	93.51	0.988	0.980	1
Hype	0.7998248	31	2	68.48	0.994	0.988	2
Crdo.s	0.8219381	22	2	56.98	0.994	0.976	2

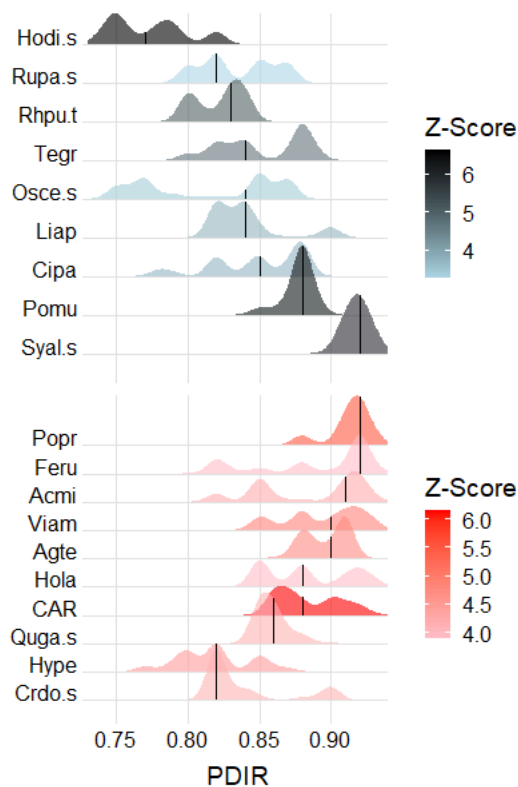
Hola	0.8496857	23	2	60.55	0.998	0.988	2
Quga.s	0.8519232	39	2	84.34	1.000	1.000	2
CAR	0.8737474	15	2	52.48	1.000	0.996	2
Viam	0.9198630	24	2	84.81	1.000	0.998	2
Popr	0.9198630	22	2	94.29	0.998	0.986	2
Agte	0.9198630	22	2	75.34	0.998	0.976	2
Feru	0.9198630	15	2	64.77	0.982	0.952	2
Acmi	0.9198630	10	2	55.67	1.000	0.956	2

The filtered species are arranged by the group they are associated with (`maxgrp`) and then by their change point (`zenv.cp`). In this case:

- Nine species were pure and reliable **decreasers** (`maxgrp` = 1) – the abundance of these species increased as PDIR declined.
- Ten species were pure and reliable **increasers** (`maxgrp` = 2) – the abundance of these species increased as PDIR increased.

We can also view the change point criteria for each species. By default, this figure focuses only on taxa that meet the purity and reliability criteria:

```
plot_taxa_ridges(titan.PDIR,
  xlabel = "PDIR")
```

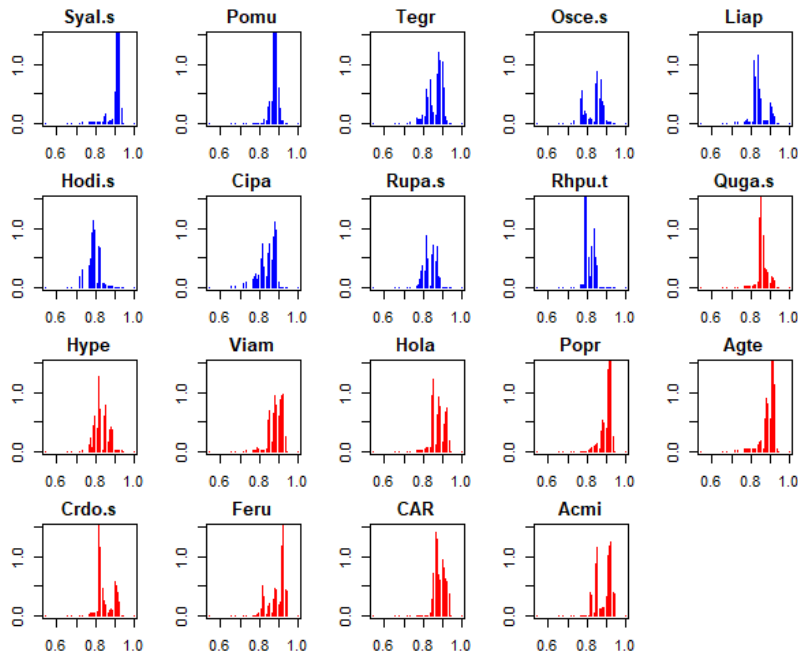


Ridge plot of species identified as strong and consistent indicators of stands with low PDIR (blue) or high PDIR (red).

Species names are identified on the y-axis. Species that are associated with smaller values (z-) or larger values (z+) of the environmental variable are shown separately and in different colors. The intensity of the color relates to the strength of the z-score obtained for IndVal. For each species, the change points show the probability density function of the bootstrapped replicates, and the vertical line is the median.

More detailed information about the change points identified through bootstrapping can be seen with the `plot_cps()` function:

```
plot_cps(titan.PDIR,
  xlabel = "PDIR")
```



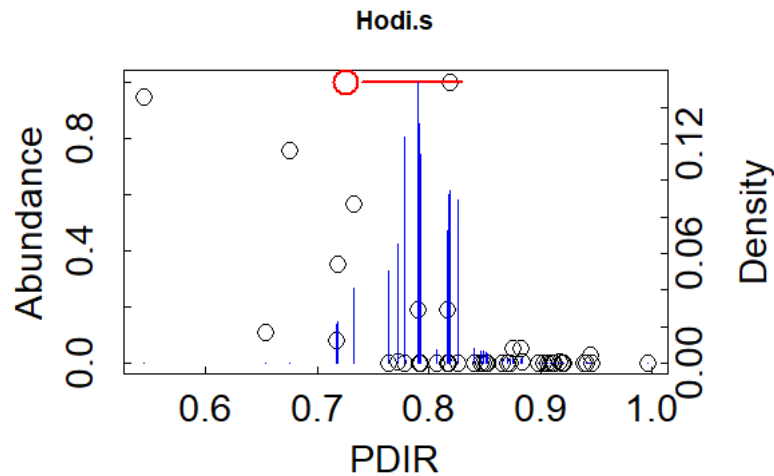
Distributions of change points below which species are strong and consistent indicators of stands with low PDIR (blue) or above which species are strong and consistent indicators of stands with high PDIR (red).

By default, this shows the change point distribution for each pure and reliable species. As is the norm in this package, decreaseers are shown in blue and increaseers in red. The distributions shown here are the same information that is shown in the ridge plot above.

A more interesting approach is to view the details for an individual species. This is done by identifying the desired species with the `taxaID` argument. (If you get an error about figure margins while running this line, clear all the plots from the plotting window first).

Let's start by focusing on `Hodi.s`, the shrub that had the lowest change point in the ridge plot:

```
plot_cps(titan.PDIR,
  taxaID = "Hodi.s",
  xlabel = "PDIR")
```



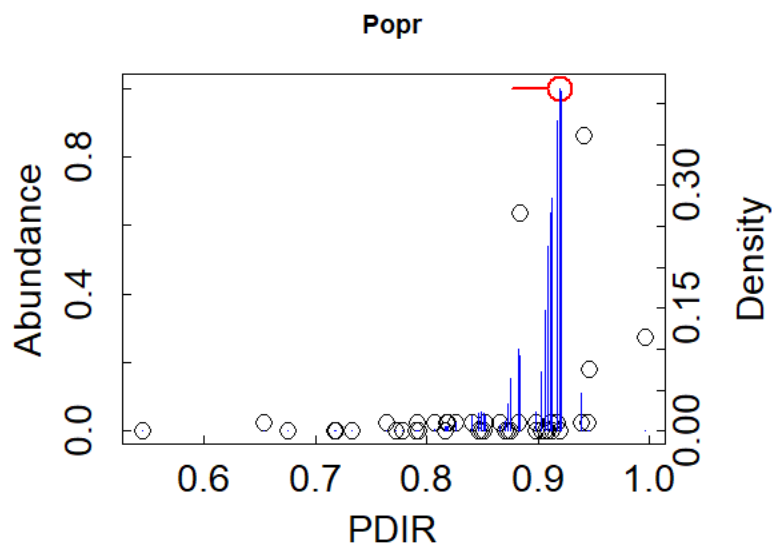
Abundance (open symbols) as a function of PDIR for Hodi.s, a decreaser. Change points identified during bootstrapping are shown by the blue bars, and the change point identified from the actual data is shown by the open red circle.

In this graphic, the black open circles are the actual abundances of this species as a function of the PDIR of each sample unit. The blue lines are the values of the change point identified in the bootstrap replicates. The large open red circle is the value of the change point identified from the actual (observed) values.

Hodi.s was more likely to be encountered in shadier stands (PDIR < ~0.72) than in sunnier stands.

For comparison, here's a graph for one of the increaser species:

```
plot_cps(titan.PDIR,
  taxaID = "Popr",
  xlabel = "PDIR")
```



Abundance (open symbols) as a function of PDIR for Popr, an increaser. Change points identified during bootstrapping are shown by the blue bars, and the change point identified from the actual data is shown by the open red circle.

Although Popr usually had low relative abundance, it was more likely to be found at higher abundance in sunnier stands (PDIR values above 0.92).

(Note: Popr is an increaser in TITAN's terminology and is shown in red when we view all species, but when viewing a single species the density distribution is shown in blue regardless of the group a species is associated with.)

Community-Level Change as a Function of PDIR

We've seen that TITAN distinguishes between species that are indicative of low values of the environmental gradient (z-; decrease) and those that are indicative of high values of the environmental gradient (z+; increase). A community-level summary of how change relates to the environmental variable is accomplished by adding up the responses of the species in each group. This can be done for all species (sumz- and sumz+ below) or for the **filtered** subset that meet the purity and reliability criteria (fsumz- and fsumz+ below).

The community-level summary is saved within the resulting object:

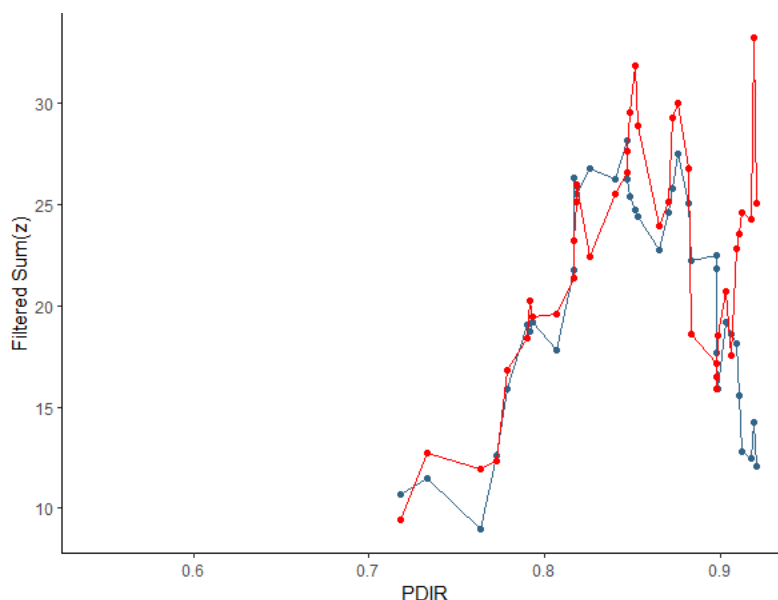
```
titan.PDIR$sumz %>%
  round(3)
```

	cp	0.05	0.10	0.50	0.90	0.95
sumz-	0.833	0.726	0.748	0.817	0.852	0.876
sumz+	0.920	0.858	0.879	0.919	0.940	0.940
fsumz-	0.847	0.800	0.817	0.847	0.879	0.883
fsumz+	0.920	0.849	0.851	0.879	0.920	0.939

The `cp` column reports the change points at which the community most strongly responded to the environmental variables. In this case, species that respond negatively to PDIR (i.e., are shade tolerant) were most often at values below ~ 0.83 whereas those that respond positively to PDIR (i.e., are shade intolerant) were most often at values above 0.92.

More details are evident graphically. Here, I use the `plot_sumz_density()` function with a few arguments to customize the graphic – see the help file for information about these and other arguments.

```
plot_sumz_density(titan.PDIR,
  ribbon = FALSE,
  points = TRUE,
  sumz = TRUE,
  change_points = FALSE,
  density = FALSE,
  xlabel = "PDIR",
  guide = "none")
```



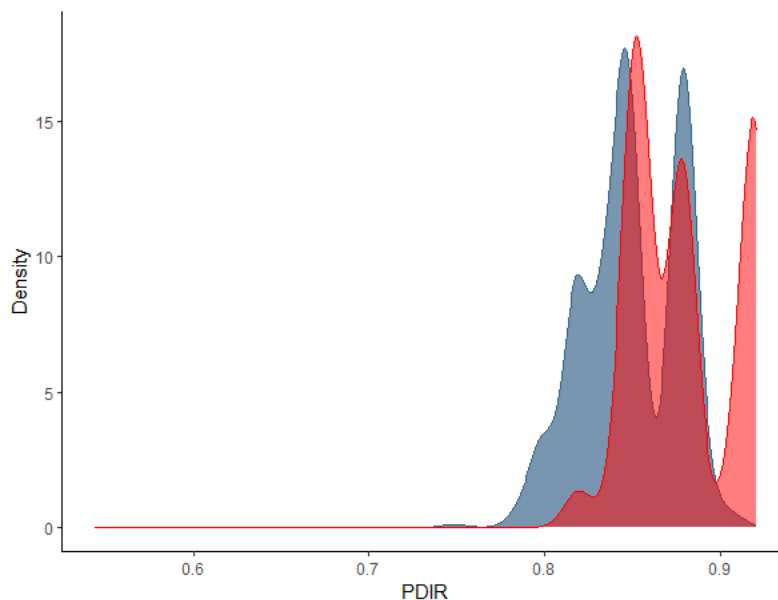
Magnitude of change for community with respect to PDIR. Blue and red lines represent the sum of the values for decrease and increase, respectively. Data are summed across all strong and consistent indicators.

This graph shows the magnitude of change for all taxa that strongly respond to PDIR (i.e., meet the purity and reliability criteria). The horizontal axis is the range of values for PDIR – each unique value was measured in one or more stands. The vertical axis is the sum of the z-scores of all species identified as pure and reliable indicators of either the z- group (decreasers; blue color) or the z+ group (increasers; red color). Peaks indicate PDIR values at which there is lots of compositional change; plateaus indicate PDIR ranges in which there is little change in composition.

PS: The above graph focused on strong responders (i.e., species that met the purity and reliability criteria). To see the community-level response based on all taxa, set `filter = FALSE`. This looks quite different!

We can use that same function to view a different graphic:

```
plot_sumz_density(titan.PDIR,  
  ribbon = FALSE,  
  points = TRUE,  
  sumz = FALSE,  
  change_points = FALSE,  
  density = TRUE,  
  xlabel = "PDIR")
```



Probability densities of change points for decrease (blue) and increase (red) as determined from bootstrap replicates.

This graph shows the probability density of change points across the bootstrap replicates. A narrow spread indicates a high degree of precision in the location of the community change point. For example, this could occur if many species are responding similarly to the same value of the environmental gradient. Usage of these density functions is left to the user – you could focus on the mode(s), for example, or the median.

Additional Applications of TITAN

In theory, many of the additional applications described for ISA can also be applied with TITAN. For example, you could conduct non-hierarchical analyses and assess species that respond to either or both of two continuous variables. Or, the outcome of a TITAN analysis of a continuous variable could be combined with the outcome of an ISA of a categorical variable.

The `pTITAN2` package allows the user to identify environmental thresholds within a site and to test whether change points vary among sites. For example, consider a site in which composition and an environmental variable such as PDIR are measured in a control area and in treated area. This is a relatively new package; I have not yet seen it in use.

Recommendations for TITAN

Many of the recommendations for using ISA and combining the results of multiple sites or studies also apply to TITAN:

1. When collecting data, identify individuals to the finest taxonomic level possible. Congeneric species, for example, may respond individually to environmental conditions.
2. Consider the proper exchangeable units for the permutations tests (Anderson & ter Braak 2003). TITAN does not allow the same level of control of permutations as is possible with ISA and many of the statistical techniques that we've discussed.
3. Direct comparisons of TITAN results from multiple studies would require that the continuous variables be measured in the same way in each study. Methods to do so should be described in detail. However, these types of comparisons are going to be more challenging than for ISA. For example, the change point differs among species and would probably differ among studies for the same species – though I'm not aware of any examples that have done this to date.
4. Ensure that the sample size is sufficient to sample the vegetation and the environmental conditions.
5. Digital appendices and other repositories permit the archiving of data in an accessible manner (Parr & Cummings 2005). Tracking species and the associated environmental variables requires more detailed data than is the case with the simplified ISA – the abundance of a species and the associated value of the environmental variable need to be recorded together as opposed to just knowing the abundance of the species and the group that each sample was part of.
6. To prevent publication bias in future meta-analyses (Gurevitch & Hedges 2001), publish data for all species, regardless of whether or not they were significant indicators.

Conclusions

TITAN is a computationally intensive amalgamation of techniques to explore how individual taxa relate to continuously-distributed variables. These variables can be measured individually or they could be derived from a multivariate analysis of a larger set of environmental variables (e.g., Costas et al. 2018).

Data adjustments remain important; these calculations will be affected by adjustments. Use the same adjustments as in all other aspects of your analysis.

References

- Anderson, M.J., and C.J.F. ter Braak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.
- Baker, M.E., and R.S. King. 2010. A new method for detecting and interpreting biodiversity and ecological community thresholds. *Methods in Ecology and Evolution* 1:25-37.
- Baker, M.E. and R.S. King. 2013. Of TITAN and straw men: an appeal for greater understanding of community data. *Freshwater Science* 32:489-506.
- Baker, M., R. King, and D. Kahle. 2020. An introduction to TITAN2. Version 2.4.3. <https://cran.r-project.org/web/packages/TITAN2/vignettes/titan2-intro.pdf>
- Beauchene, M., M. Becker, C.J. Bellucci, N. Hagstrom, and Y. Kanno. 2014. Summer thermal thresholds

- of fish community transitions in Connecticut streams. *North American Journal of Fisheries Management* 34(1):119-131.
- Costas, N., I. Pardo, L. Méndez-Fernández, M. Martínez-Madrid, and P. Rodríguez. 2018. Sensitivity of macroinvertebrate indicator taxa to metal gradients in mining areas in Northern Spain. *Ecological Indicators* 93:207-218.
- Farwell, L.S., P.B. Wood, R. Dettmers, and M.C. Brittingham. 2020. Threshold responses of songbirds to forest loss and fragmentation across the Marcellus-Utica shale gas region of central Appalachia, USA. *Landscape Ecology* 35: 1353-1370.
- Gurevitch, J., and L.V. Hedges. 2001. Meta-analysis: combining the results of independent experiments. Pages 347-369 in S.M. Scheiner and J. Gurevitch (editors). *Design and analysis of ecological experiments*. Oxford University Press, New York, NY.
- King, R.S., and M.E. Baker. 2010. Considerations for analyzing ecological community thresholds in response to anthropogenic environmental gradients. *Journal of the North American Benthological Society* 29:998-1008.
- Martin, R.A., and S.T. Hamman. 2016. Ignition patterns influence fire severity and plant communities in Pacific Northwest, USA, prairies. *Fire Ecology* 12:88-102.
- McBurney, K.G., E.T. Cline, J.D. Bakker, and G.J. Ettl. 2017. Ectomycorrhizal community composition and structure of a mature red alder (*Alnus rubra*) stand. *Fungal Ecology* 27:47-58.
- Morissette, J.L., E.M. Bayne, K.J. Kardynal, and K.A. Hobson. 2019. Regional variation in responses of wetland-associated bird communities to conversion of boreal forest to agriculture. *Avian Conservation and Ecology* 14(1):12.
- Parr, C.S., and M.P. Cummings. 2005. Data sharing in ecology and evolution. *Trends in Ecology and Evolution* 20:362-363.
- Smucker, N.J., E.M. Pilgrim, C.T. Nietch, J.A. Darling, and B.R. Johnson. 2020. DNA metabarcoding effectively quantifies diatom responses to nutrients in streams. *Ecological Applications* 30:e02205.

Media Attributions

- PDIR
- TITAN.PDIR.ridges
- TITAN.PDIR.cps
- TITAN.PDIR.Hodi.s
- TITAN.PDIR.Popr
- TITAN.PDIR.sumz
- TITAN.PDIR.density

Appendix 1: Order of Data Adjustments

McCune & Grace (2002) provided a very helpful summary of the order in which data adjustments should be made in Tables 9.3 (species data) and 9.4 (environmental data). Those tables are summarized, in a slightly updated form, here.

Species Data

This assumes that the data are in a matrix with sample units as rows and species data as columns.

Action to be considered	Criteria										
1. Calculate descriptive statistics (beta diversity, average skewness of columns, CV of row totals, CV of column totals). Repeat after each step below.	Always										
2. Delete rare species (<5% of sample units)	Unless contrary to study goals Consider:										
3. Monotonic transformation. If applied to species, usually applied uniformly to all of them so that all are scaled the same.	<ul style="list-style-type: none"> · Average skewness of columns · Data range: how many orders of magnitude? · Beta diversity <p>Appropriateness depends on the question being addressed. Regardless of whether you relativize or not, you should briefly state and justify your decision.</p>										
4. Row or column relativizations	<p>Are units for all variables the same?</p> <p>Is relativization built into distance measure or subsequent analyses?</p> <p>CV of row totals</p> <p>CV of column totals</p> <table> <tr> <th>SD</th><th>Degree of problem</th></tr> <tr> <td><2</td><td>No problem</td></tr> <tr> <td>2-2.3</td><td>Weak outlier</td></tr> <tr> <td>2.3-3</td><td>Moderate outlier</td></tr> <tr> <td>>3</td><td>Strong outlier</td></tr> </table>	SD	Degree of problem	<2	No problem	2-2.3	Weak outlier	2.3-3	Moderate outlier	>3	Strong outlier
SD	Degree of problem										
<2	No problem										
2-2.3	Weak outlier										
2.3-3	Moderate outlier										
>3	Strong outlier										
5. Check for outliers based on average distance of each point from all other points. Calculate standard deviation of these average distances. Describe outliers and take steps to reduce influence, if necessary.											

Environmental Data

This assumes that the data are in a matrix with sample units as rows and quantitative environmental data as columns.

Action to be considered	Criteria
1. Calculate descriptive statistics (skewness and range for each column). Repeat after each step below.	Always
2. Monotonic transformation. Apply to individual variables (columns) depending on need.	<p>Consider log or square root transformation for variables with skewness > 1 or spanning several orders of magnitude.</p> <p>Consider arcsine square root transformation for proportion data.</p> <p>Consider column relativization (e.g., by norm or standard deviate) if environmental variables are to be used in a distance-based analysis that does not automatically relativize the variables.</p>
3. Column relativizations	Not necessary for analyses that use the variables one at a time or for analyses with built-in standardization (e.g., PCA of a correlation matrix).
4. Check for univariate outliers and take corrective steps if necessary.	Examine scatterplots or frequency distributions, or relativize by standard deviate and check for high absolute values.

Appendix 2: Structure of Complex Experimental Designs

Somerfield (2021a, b, c) published a series of related articles explaining how ANOSIM can be applied to a variety of designs. I reprint some of their key tables here.

Designs are provided for:

- Crossed or nested designs
- Ordered or unordered factors. For example, high / medium / low would be ordered whereas prairie / forest / wetland would be unordered.
- Replication at the lowest level of the design

These tables were prepared for and focus on ANOSIM, but elements such as how permutations need to be restricted for each term are also relevant for statistical techniques such as PERMANOVA and RRPP. Also included (right-hand column) are some ecological examples that illustrate possible usages of each design.

Additional information about how to analyze complex designs is provided by Anderson & ter Braak (2003) – see in particular Tables A1 through A1V in their Appendix.

1-Factor and 2-Factor Designs

Table 1 from Somerfield et al. (2021b) provides the following guidance for using ANOSIM to analyze 1-factor (options 1a to 1d) and 2-factor (options 2a to 2n) experimental designs using the generalized ANOSIM test statistic.

The table is in two parts as it spans two pages in the original publication.

Table 1. 1-way and 2-way ANOSIM (global) test statistics, for crossed and nested designs, with unordered or ordered factors, and with or without replication at the lowest level of the design. Also given are the possibility (or not) of pairwise tests, details of the test constructions and examples of contexts in which they might be employed

No.	Type of design	Factor(s)	Factor level ordering	Replicates?	Statistics used	Pairwise test?	Construction of statistic	Examples
1a	1-way	A	Unordered	Yes	R	Yes	A: Standard 1-way ANOSIM statistic [‡]	A: sites, with replicates in each
1b	1-way	A	Unordered	No	-	-	A: No basis for a test	-
1c	1-way	A	Ordered	Yes	R^{Oc}	Yes	A: ANOSIM form of seriation statistic for ordered categories [§]	A: impact levels, expecting monotonic response
1d	1-way	A	Ordered	No	R^{Ox}	No	A: ANOSIM form of simple seriation statistic (no replicates) [§]	A: inter-annual trend or positions along a transect
2a	2-way crossed	AxB	A unordered	Yes	A: \bar{R}	Yes	A: Average of 1-way R for testing A across separate levels of B	A: shores, B: treatment types (several applications), or
			B unordered		B: \bar{R}	Yes	B: Average of 1-way R for testing B across separate levels of A	A: locations, B: habitats (sites as replicates)
2b	2-way crossed	AxB	A unordered	No	A: ρ_{ao}	No	A: Average of ρ among resemblance matrices (of A) across levels of B [¶]	As 2a but each treatment only once on each shore, or
			B unordered		B: ρ_{av}	No	B: Average of ρ among resemblance matrices (of B) across levels of A [¶]	A: sites, B: times, each site visited once at each time
2c	2-way crossed	AxB	A unordered	Yes	A: \bar{R}	Yes	A: As test 2a	A: shores, B: increasing treatment impact levels, or
			B ordered		B: \bar{R}^{Oc}	Yes	B: Average of 1-way R^{Oc} for testing B across separate levels of A	A: locations, B: water depths (sites as replicates)
2d	2-way crossed	AxB	A unordered	No	A: ρ_{ao}	No	A: As test 2b	A: site, B: tidal height (transect down shore) or
			B ordered		B: \bar{R}^{Ox}	No	B: Average of 1-way R^{Ox} for testing B across separate levels of A	A: patch reefs, B: inter-annual trend
2e	2-way crossed	AxB	A ordered	Yes	A: \bar{R}^{Oc}	Yes	A: Average of R^{Oc} for testing A across B levels (i.e. 2c, switching A and B)	A: shores on latitudinal gradient,
			B ordered		B: \bar{R}^{Oc}	Yes	B: As 2c	B: coarseness of sediment classes, replicate sites in each combination
2f	2-way crossed	AxB	A ordered	No	A: \bar{R}^{Ox}	No	A: Average of R^{Ox} for testing A across B levels (i.e. 2d, switching A and B)	A: transect of sites along shore and
			B ordered		B: \bar{R}^{Ox}	No	B: As 2d	B: depth transect at each site, sampling (once) the same set of depths
2g	2-way nested (B within A)	B(A)	A unordered	Yes	A: R	Yes	A: As test 1a, but with levels of B as replicates (averaging within those)**	A: protected/not protected areas,
			B unordered		B: \bar{R}	No	B: As test 2a, but without pairwise tests**	B: sites within each type (replicates are trawls within each site)
2h	2-way nested	B(A)	A unordered	No	A: R	Yes	A: As test 1a, but this time the sole levels of B are the only replicates	A: location,
			B unordered		B: -	-	B: No basis for a test	B: site (e.g. time-averaged to give one sample for each site)
2i	2-way nested	B(A)	A ordered	Yes	A: R^{Oc}	Yes	A: As test 1c, but with levels of B as replicates (averaging within those)**	A: depth bands,
			B unordered		B: \bar{R}	No	B: As test 2g	B: random sites in each depth band, replicate grab samples at each site

Table 1. *Continued*

No.	Type of design	Factor(s)	Factor level ordering	Replicates?	Statistics used	Pairwise test? [†]	Construction of statistic	Examples
2j	2-way nested	B(A)	A ordered	No	A: R^{Oc}	Yes	A: As test 1c, but this time the sole levels of B are the only replicates	A: distance from outfall,
			B unordered		B: -	-	B: No basis for a test	B: random sites at each distance, and 'pseudo-replicates' (e.g. multicorer) pooled
2k	2-way nested	B(A)	A unordered	Yes	A: R	Yes	A: As test 2g (ordered levels of B assumed representative as replicates) ^{**}	A: dry/wet season,
			B ordered		B: \overline{R}^{Oc}	No	B: As test 2c, but without pairwise tests ^{**}	B: months (replicates as random days in month)
2l	2-way nested	B(A)	A unordered	No	A: R	Yes	A: As test 2h (ordered levels of B assumed representative as replicates) ^{**}	A: site,
			B ordered		B: \overline{R}^{Os}	No	B: As test 2d	B: points along transect (one transect at each site, randomly oriented and located)
2m	2-way nested	B(A)	A ordered	Yes	A: R^{Oc}	Yes	A: As test 2i (ordered levels of B assumed representative as replicates)	A: region, latitudinally arranged,
			B ordered		B: \overline{R}^{Oc}	No	B: As test 2k	B: transect of sites in each region (all at same depth), replicates within
2n	2-way nested	B(A)	A ordered	No	A: R^{Oc}	Yes	A: As test 2j (ordered levels of B assumed representative as replicates)	A: seamounts in different depth classes,
			B ordered		B: \overline{R}^{Os}	No	B: As test 2l	B: distance along single random transect on each seamount

[†]All pairwise tests are unordered, by definition.

[‡] $R = (\overline{r}_B - \overline{r}_W)/(M/2)$, equivalently the slope of a linear regression of ranks of the biotic resemblances against ranks from a (0,1) model for levels of A.

[§] R^{Oc} is the slope from a linear regression of ranks of biotic resemblances against ranks from a 'seriation with replication' model matrix and R^{Os} against a simple seriation model without replication; they are the (asymmetric) ANOSIM R forms of the (symmetric) RELATE Spearman ρ statistic. The distinction between ordered categories (R^{Oc}) and simple seriation (R^{Os}) is not crucial for calculation purposes (thus R^O).

^{||}Matrix correlation (Spearman rank ρ) calculated between all pairs of biotic resemblance matrices (for levels of A) within levels of B, and then ρ averaged over the separate B levels to give ρ_{av} for A (vice-versa for B).

^{**}Ranked resemblances are averaged within levels of B(A), and for all pairs across levels of B(A); the resulting averaged matrix is re-ranked and input to 1-way ANOSIM for levels of A, using B levels as replicates. The same is done for each of the pairwise tests, first selecting only resemblances for the requisite pair of A levels, then ranking, averaging and re-ranking before inputting the two levels to 1-way ANOSIM.

^{††}The global test is the same as the crossed case but here the levels of B, even if similarly denoted (by 1, 2, ... say) have nothing in common across the levels of A, so a pairwise test of B1 v B2 (say) is meaningless.

^{‡‡}A nested factor might typically be a randomly located site (B) in a region (A). Ordered sites might come from transects of sites across each region (randomly directed so transect points are nested not crossed with region). If representative of the region's extent, transect sites could still be considered suitable replicates for a test of region, the 'randomness' coming from the stochastic nature of the environment being sampled.

3-Factor Designs

Somerfield et al. (2021c) provides the following guidance for using ANOSIM to analyze 3-factor (options 3a to 3m) experimental designs using the generalized ANOSIM test statistic.

The table is in two parts as it spans two pages in the original publication.

Table 1. 3-way ANOSIM (global) test statistics, for crossed and nested designs, with unordered or ordered factors, and with or without replication at the lowest level of the design

No.	Type of design	Factors	Factor levels ordered?	Replicates?	Statistics used	Pairwise test?	Construction of test	Examples
3a	3-way crossed	A×B×C	A, B, C unordered	Yes	A, B, C: \bar{R}	Yes	As two-way crossed test, but combining pairs of factors in turn, for example calculating 1-way R for A within all B×C levels [†]	A: location, B: time, C: habitat
3b	3-way crossed	A×B×C	A, B, C unordered	No	A, B, C: ρ_{av}	No	As two-way crossed test with no replication, that is comparing resemblance matrices of A across combined B×C levels [†]	As 3a above but no reps (or pooled)
3c	3-way crossed	A×B×C	A, B unordered C ordered	Yes/no	A, B: \bar{R} C: $\bar{R}^{Oc}/\bar{R}^{Oa}$	Yes/no	A, B: as test 3a/3b C: as 2-way crossed test, collapsing A, B to single factor A×B [†]	A: location, B: time, C: depth range with/without reps in A×B×C cells
3d	3-way nested, C within B within A	C(B(A))	A, B, C unordered	Yes	A: R B, C: \bar{R}	A: Yes B, C: No	A, B: as 2-way nested test of B in A, using levels of C as replicates [‡] C: as 2-way nested test for C in all B levels (i.e. over all A levels)	A: region, B: location, C: site, with replicate samples at each site
3e	3-way nested, C within B within A	C(B(A))	A, B, C unordered	No	A: R B: \bar{R} C: —	A: Yes B: No C: —	A, B: exactly as for test 3d (except no averaging of C level reps needed) C: no basis for a test	A: region, B: location, C: site, with one pooled sample at each site
3f	3-way nested, C within B within A	C(B(A))	A, B unordered C ordered	Yes/no	A: R B: \bar{R} C: $\bar{R}^{Oc}/\bar{R}^{Oa}$	A: Yes B, C: No	A, B: as 2-way test of B nested in A, using ordered C levels (single C values) as reps [‡] C: as 2-way ordered test of C nested in B(A), all B levels over A	A: location, B: shore, C: along shore transect, reps (or not) at transect points
3g	3-way nested, C within B within A	C(B(A))	A unordered B ordered, C either	Yes/no	A, C: as 3f B: \bar{R}^{Oc}	A: Yes B, C: No	A, C: as the relevant tests in 3d–3f B: as 2-way nested test for B (ordered) within A, using levels of C (single C values) as reps [‡]	A: sea region, B: transect of sites, C: random days at each site (with/without rep trawls)
3h	3-way, C nested in A×B	C(A×B)	A, B, C ordered or unordered	Yes/no	Various	A, B: Yes C: No	A, B: as for 2-way crossed tests but using C levels as reps (averaged where needed) ^{††} C: as for 2-way nested, C in all combinations A×B	A: location, B: season, C: different site-day combinations in each A×B (with/without rep. cores)
3i	3-way, B crossed with C(A) (i.e. only C is nested in A)	B×C(A)	A, B, C unordered	Yes	A: \bar{R} B: \bar{R} C: \bar{R}	A: Yes B: Yes C: No	A: average the reps in C levels (on resemblances [‡]), then 2-way crossed statistic for A from A×B ^{‡‡} B: usual 2-way crossed test for B across all levels of C (over all A) C: usual 2-way nested test for C within all combined levels A×B	A: location, B: time, C: same random sites in location returned to each time, with replicate samples at sites
3j	3-way, B crossed with C(A)	B×C(A)	A, B, C unordered	No	A: \bar{R} B: $\bar{\rho}_{av}$ C: $\bar{\rho}_{av}$	A: Yes B: No C: No	A: as 3i but with single C levels as reps (constrained perms again ^{††}) B: ρ_{av} statistic for B patterns matched over C levels in each A, then averaged (normal perms) ^{‡‡} C: converse ρ_{av} of C patterns, for each A, matched across B levels, then ρ_{av} averaged over A ^{††}	A: location, B: time, C: same random sites in location returned to each time for single sample (or pooled sample)

Table 1. *Continued*

No.	Type of design	Factors	Factor levels ordered?	Replicates?	Statistics used	Pairwise test?	Construction of test	Examples
3k	3-way, B crossed with C(A)	BxC(A)	A unordered B unordered C ordered	Yes/no	A: \bar{R} B: $\bar{R}/\bar{\rho}_{av}$ C: $\bar{R}^{\alpha_c}/\bar{R}^{\alpha_r}$	A: Yes B: Yes/no C: No	A: as test 3i/3j B: as test 3i/3j C: as 2-way test of C(AxB), that is C nested in all AxB combinations	A: location, B: time, C: same (representative) transect of sites in location returned to each time
3l	3-way, B crossed with C(A)	BxC(A)	B ordered A,C ordered or unordered	Yes/no	B: $\bar{R}^{\alpha_c}/\bar{R}^{\alpha_r}$ A,C: as 3i-k or 3m	B: Yes/no A: Yes, C: No	B: 2-way crossed test of ordered B with all levels of C (in all A) A,C: as the relevant tests for A,C in 3i-kjm	A: location, B: yearly time trend, C: same random sites in each location each year
3m	3-way, B crossed with C(A)	BxC(A)	A ordered B,C ordered or unordered	Yes/no	A: \bar{R}^{α_c} B,C: as 3i-l	A: Yes B: Yes/no C: No	A: 2-way crossed test of ordered A across B, using C levels as reps (C reps if present are averaged ^b), B levels held as a block in perms ^{††} B,C: as the relevant tests in 3i-3l	A: latitudinal region, B: yearly trend, C: same transect of sites in region each year (with/ without reps). A, B,C ordered

Also given are the existence (or not) of pairwise tests, details of the test constructions and examples of contexts in which they might be employed.

[†]Test for A uses average of 1-way R (for A) across all levels of B and C in combination ($B \times C$), then $B \vee (A \times C)$ and $C \vee (A \times B)$. Same idea is used for 3b (use the 2-way test for unreplicated designs), and if two of the factors are ordered still use 3a, b or c.

[‡]Starts from ranked resemblances of C replicates, which are then averaged and re-ranked (twice for the A test). Or (e.g. if unsure of quality of C replicates) test A & B by averaging C replicates in data matrix and using a 2-way test on A and B(A).

[§]C levels (averaged where needed, as in note ‡) are assumed representative replicates of B(A) condition.

[¶]If A ordered (whether B, C are or not), it changes nothing except the test of A, in which ordered levels of B are now assumed representative as replicates.

^{††}Similar comments as for note ‡ apply, about whether it may be better sometimes to average replicates of C externally, on the data matrix, then calculate resemblances and submit to the 2-way crossed cases for $A \times B$.

^{‡‡}Note the necessity for a block-constrained permutation test here under the null, with values across B for each C level being permuted as a batch across C(A) and A levels. A common structure is A: locations, C: sites (nested in A), B: period, all sites visited in each period. Test for A uses sites as replicates but keeps the periods for each site together under permutation across locations.

^{§§}This is a new doubly averaged statistic $\bar{\rho}_{av}$ matching patterns in B over the C levels for each A level (the usual ρ_{av}), then averaging ρ_{av} over A levels. Permutations are the usual random ordering of B for each C(A).

^{¶¶}For example ρ_{av} calculated to match relationships among sites for different periods, separately for each location, then ρ_{av} averaged over locations. Standard permutation of sites within all levels of location \times period.

References

- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021a. A generalized analysis of similarities (ANOSIM) statistic for designs with ordered factors. *Austral Ecology* 46:901-910.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021b. Analysis of similarities (ANOSIM) for 2-way layouts using a generalized ANOSIM statistic, with comparative notes on Permutational Multivariate Analysis of Variance (PERMANOVA). *Austral Ecology* 46:911-926.
- Somerfield, P.J., K.R. Clarke, and R.N. Gorley. 2021c. Analysis of similarities (ANOSIM) for 3-way designs. *Austral Ecology* 46:927-941.

Media Attributions

- Somerfield.2way.Table1a
- Somerfield.2way.Table1b
- Somerfield.3way.Table1a
- Somerfield.3way.Table1b

Appendix 4: Contrasts

Learning Objectives

To learn how to build contrasts in PERMANOVA and other R functions.

Contents:

- General Principles
- A More Detailed Grazing Example
- Pairwise Contrasts
- Different Methods Yield Different Conclusions
- Custom Contrasts
- References

In the chapter about complex models, I presented a simple way to do pairwise contrasts. Here, I want to describe some more general concepts related to contrasts. I present these in the context of PERMANOVA, but they are general concepts— they are applicable for all linear models in R, including conventional parametric tests such as `lm()` and `aov()`. After outlining some general principles, we will consider how to do pairwise contrasts and custom contrasts.

General Principles

R, like all statistical packages, deals with factors via **contrasts**. Contrasts are an important follow-up to many analyses. Some – but not all – analytical techniques have pre-defined functions that can be used to conduct pair-wise comparisons. However, you can also specify these comparisons yourself for complete control over your analyses.

You may recall from parametric statistics courses that the analysis of a factor with n groups can be decomposed into $n-1$ contrasts. Crawley (2012) provides a very good explanation and demonstration of contrasts. He also demonstrates an approach in which the model is simplified by aggregating non-significant factor levels in a step-wise *a posteriori* procedure.

Each contrast consists of a series of coefficients with the following characteristics:

- Levels that are being combined get the same sign
- Levels that are being contrasted get opposing signs
- Levels that are being omitted get a coefficient of 0

When using contrasts, it is often helpful that they be **orthogonal** (i.e., uncorrelated with one another). Using orthogonal contrasts enables you to fully decompose the variability associated with a factor into the variability associated with each contrast. In addition to the three characteristics listed above, orthogonal contrasts also have the following characteristics:

- The coefficients in a given contrast sum to zero
- The cross-products of any two contrasts sum to zero

A key principle when conducting a contrast is to pay attention to how variance is being partitioned. An analysis of a factor with two levels obviously does not require follow-up contrasts, so our focus is on factors with > 2 levels and thus > 1 df. Our intent is to fully partition the df and variance (SS) associated with the factor into the df and SS associated with the contrast of interest and the df and SS associated with other aspects of the factor. If these are not fully partitioned, the unaccounted-for df and SS will be lumped into the residual. Depending on their magnitude, this may impact conclusions about the statistical significance of the contrast of interest.

A More Detailed Grazing Example

The oak plant community dataset contains factors indicating whether stands were grazed in the past and/or grazed at the time the data were collected (`GrazPast` and `GrazCurr`, respectively). We'll combine these factors together into a new `Grazing` factor (capitalized):

```
Grazing <- factor(with(Oak, paste(GrazPast, GrazCurr, sep = "_"))
summary(Grazing)
```

```
No_No  Yes_No Yes_Yes
  24      6     17
```

Since each of the original factors is binary, there are four possible levels here. However, there are no stands that were not grazed in the past but were being grazed at the time the data were collected (i.e., the 'No_Yes' combination is absent).

Let's test the overall effect of `Grazing` on vegetation composition, as measured in `Oak1`:

```
Graz.res2 <- adonis2(Oak1 ~ Grazing, method = "bray")
```

View ANOVA table (`Graz.res2`):

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ Grazing, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
Grazing  2   0.9095 0.07844 1.8725  0.006 **
Residual 44  10.6854 0.92156
Total   46  11.5949 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Yes, there is a relationship between Grazing and composition. But which levels differ from one another? To determine this, we need to use pairwise contrasts.

There are several ‘families’ of contrasts that can be applied in linear models. For our purposes here, we will focus on two:

- `contr.treatment` – sets the first level to be the baseline and contrasts each other level with the baseline. The help file notes that this contrast may not be orthogonal.
- `contr.sum` – contrasts levels directly against each other.

The desired family can be specified using the `C()` function (note: capital C!):

`C(Grazing, contr.sum)`

```
[1] Yes_Yes Yes_No Yes_Yes No_No No_No Yes_Yes Yes_Yes
[8] No_No No_No No_No Yes_Yes Yes_Yes Yes_Yes Yes_Yes
[15] Yes_No Yes_No No_No No_No No_No No_No No_No
[22] No_No No_No No_No No_No No_No No_No No_No
[29] Yes_Yes No_No No_No No_No Yes_Yes No_No Yes_Yes
[36] Yes_Yes Yes_Yes Yes_No Yes_No Yes_Yes Yes_Yes Yes_Yes
[43] Yes_Yes No_No Yes_No No_No No_No
attr(,"contrasts")
      [,1] [,2]
No_No      1      0
Yes_No      0      1
Yes_Yes     -1     -1
Levels: No_No Yes_No Yes_Yes
```

The contrasts that R applies in an analysis are stored in a model matrix. Note that the function to create this matrix uses a formula:

`Grazing.mm <- model.matrix(~ C(Grazing, contr.sum))`

```
(Intercept) C(Grazing, contr.sum)1 C(Grazing, contr.sum)2
1           1           -1           -1
2           1            0            1
3           1           -1           -1
4           1            1            0
5           1            1            0
6           1           -1           -1
7           1           -1           -1
8           1            1            0
9           1            1            0
10          1            1            0
11          1           -1           -1
12          1           -1           -1
13          1           -1           -1
14          1           -1           -1
15          1            0            1
```

```

16          1          0          1
17          1          1          0
18          1          1          0
19          1          1          0
20          1          1          0
21          1          1          0
22          1          1          0
23          1          1          0
24          1          1          0
25          1          1          0
26          1          1          0
27          1          1          0
28          1          1          0
29          1         -1         -1
30          1          1          0
31          1          1          0
32          1          1          0
33          1         -1         -1
34          1          1          0
35          1         -1         -1
36          1         -1         -1
37          1         -1         -1
38          1          0          1
39          1          0          1
40          1         -1         -1
41          1         -1         -1
42          1         -1         -1
43          1         -1         -1
44          1          1          0
45          1          0          1
46          1          1          0
47          1          1          0
attr(,"assign")
[1] 0 1 1
attr(,"contrasts")
attr(,"contrasts")$`C(Grazing, contr.sum)`
      [,1] [,2]
No_No    1    0
Yes_No    0    1
Yes_Yes  -1   -1

```

By examining this object, we can see that:

- The first column is for the intercept
- The attributes at the bottom show that the first contrast compares 'No_No' (1) with 'Yes_Yes' (-1), ignoring 'Yes_No' (0)
- The second contrast compares 'Yes_No' (1) with 'Yes_Yes' (-1), ignoring 'No_No' (0)

Note: For more information about contrasts, see the R help files and Crawley (2012).

Pairwise Contrasts

Pairwise contrasts require separate tests of all possible pairs of levels. The number of pairs of n levels is the same as the number of distances from n sample units: $n(n-1)/2$.

In this example, there are 3 levels so there are $3(3-1)/2 = 3$ pairs. We are interested in determining, for each pair, whether there is a significant difference between them. We can organize the results in a table (not shown) or a distance matrix like this:

	No_No	Yes_No
Yes_No		
Yes_Yes		

For convenience, we are just going to display the P -values.

In essence, what we are going to do is test each pair while accounting for the other variation that is attributable to the grazing factor. Our grouping variable had 3 levels, so there were 2 df. This means that in each test we can test no more than two pairwise contrasts. And, if we fit a single pairwise contrast we will need to make sure to also account for the other df and its associated variation.

The model matrix that we saw above includes two pairwise contrasts. If we fit the full model matrix as a single explanatory variable, we get the same result as when we used the Grazing term above:

```
adonis2(Oak1 ~ Grazing.mm, method = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ Grazing.mm, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
Grazing.mm  2    0.9095 0.07844 1.8725  0.004 **
Residual   44   10.6854 0.92156
Total      46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

However, by treating each pairwise contrast as its own explanatory variable, we can partition the variance to these terms. We could do this either by indexing the model matrix within the `adonis2()` function or by creating a new object for each contrast like this:

```
con.NNvYY <- Grazing.mm[, 2]
con.YNvYY <- Grazing.mm[, 3]
```

If we only use one of these contrasts, the variance is not partitioned correctly:

```
adonis2(Oak1 ~ con.YNvYY, method = "bray")
```



```

Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ con.YNvYY, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
con.YNvYY  1      0.317 0.02734 1.2651 0.211
Residual 45     11.278 0.97266
Total     46     11.595 1.00000

```

Only 1 df has been used to test for the grazing effect. The other df and its associated variance have been included in the residual. (I used `con.YNvYY` here as it illustrates this point more strongly than `con.NNvYY`).

Note: this is different from the pairwise comparisons that we conducted earlier (`pairwise.adonis2()`); in that case we subsetting the data to a pair of levels. Stands with the other grazing treatment were excluded from the analysis, so their variance was not included in the residual.

Substitute both contrasts into the function, and verify that the df and SS for grazing have been fully partitioned between them:

```
adonis2(Oak1 ~ con.NNvYY + con.YNvYY, method = "bray")
```

```

Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ con.NNvYY + con.YNvYY, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
con.NNvYY  1      0.7869 0.06787 3.2404 0.001 ***
con.YNvYY  1      0.1225 0.01057 0.5046 0.981
Residual 44     10.6854 0.92156
Total     46     11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Note that the residual df and SS are identical to what they were when we used `Grazing` or `Grazing.mm` (each with three levels of this factor).

However, we are now dealing with a more complex model than we have seen previously: this model includes two terms, and is unbalanced. This means that the order in which terms enter the model matters – see the discussion of types of sums of squares in the ‘Complex Models’ chapter. The first contrast is accurate because it is the first term in the model, but the second one may not be.

The conclusions about a term depend on where it is in the formula, and on how other terms are dealt with. When conducting contrasts, we want each term to explain as much variation as possible. To

do so, we will enter the contrast of interest as the first term in the model and the other contrast as the second term so that it accounts for as much of the remaining variation as possible. We could also specify `by = "terms"` for completeness, though this is not strictly necessary as it is the default approach in `adonis2()`.

Here is the model with our two contrasts reversed so that `con.YNvYY` is in the first position:

```
adonis2(Oak1 ~ con.YNvYY + con.NNvYY, method = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ con.YNvYY + con.NNvYY, method = "bray")
              Df SumOfSqs      R2      F Pr(>F)
con.YNvYY     1    0.3170 0.02734 1.3055  0.163
con.NNvYY     1    0.5924 0.05109 2.4394  0.002 **
Residual     44   10.6854 0.92156
Total        46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the *P*-value for `con.YNvYY` has changed considerably. In this case, we still conclude that there is no significant difference in composition between the 'Yes_No' and 'Yes_Yes' grazing treatments. However, it is possible for one analysis to indicate significance and the other to indicate non-significance.

We can enter both of these pairwise contrasts in our matrix:

	No_No	Yes_No
Yes_No		
Yes_Yes	0.001	0.163

Now, what about the last pairwise contrast, between 'No_No' and 'Yes_No'? To test this, we need to create a new contrast. We will do so by following four simple steps:

1. Create a duplicate of the original factor. This is not strictly necessary, but I do this so that my original object is unchanged and I can easily refer back to it if desired.
`Grazing1 <- Grazing`
2. Designate our own contrasts for this term. Contrasts are applied to the levels of a factor in the order in which they are summarized when you view them; the default is to report them alphabetically. See `levels(Grazing1)` to verify. Since we want to contrast 'No_No' with 'Yes_No', our coefficients will be (1, -1, 0). Two contrasts are possible among these three levels; if we do not specify the second, it will be created automatically to be orthogonal to the one we specified. We will apply this contrast to a new model matrix:
`Grazing.mm1 <- model.matrix(~ C(Grazing1, c(1, -1, 0)))`
Verify that the contrasts have been applied to the levels of the factor as intended. For example,

every row that belongs to the 'No_No' level is assigned a '1' in the first contrast.

3. Use indexing to create a separate object for each contrast.

```
con.NNvYN <- Grazing.mm1[, 2]
```

```
con.NNvYN.rest <- Grazing.mm1[, 3]
```

Note that the last contrast here is one that we may not be interested in but need to include for completeness.

4. Rerun our analysis with the new contrasts, specifying the 'focal' contrast as the first one in the model.

```
adonis2(Oak1 ~ con.NNvYN + con.NNvYN.rest, method = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ con.NNvYN + con.NNvYN.rest, method = "bray")
              Df SumOfSqs      R2      F Pr(>F)
con.NNvYN      1   0.5389 0.04647 2.2189  0.002 **
con.NNvYN.rest 1   0.3706 0.03196 1.5260  0.066 .
Residual      44  10.6854 0.92156
Total         46  11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Verify once again that the df and variance have been fully partitioned between the contrasts.

Finally, we update our matrix of pairwise contrasts:

	No_No	Yes_No
Yes_No	0.002	
Yes_Yes	0.001	0.163

Now that we have completed the table, we can inspect the pairwise contrasts and summarize what they mean ecologically. Here, we see evidence that composition differs between stands that have never been grazed ('No_No') and those that have been grazed at any point. Stands that were grazed in the past do not differ in composition, regardless of whether they were being grazed at the time the data were collected.

Different Methods Yield Different Conclusions

You may be interested in seeing how well the above matrix of pairwise comparisons agrees with the matrix of comparisons we created in the 'Complex Models' chapter. That one is repeated here:

	No_No	Yes_No
Yes_No	0.359	
Yes_Yes	0.001	0.932

These results obviously do not match. Why not? There are several possible reasons:

- Using a different set of permutations for each test
- Data are unbalanced
- When using custom contrasts as in this appendix, the inclusion of the other level of the grazing factor may influence the residual (variation within levels of grazing)

Some exploration suggests that the last of those reasons is the main 'culprit'. For example, consider what happens when we index the response and explanatory objects to only include two levels of grazing and do the PERMANOVA on that reduced dataset:

```
g2 <- Grazing %in% c("Yes_No", "Yes_Yes")
set.seed(42)
adonis2(Oak1[g2, ] ~ Grazing[g2], method = "bray")
# F = 0.58; P = 0.943
```

```
g2 <- Grazing %in% c("Yes_No", "No_No")
set.seed(42)
adonis2(Oak1[g2, ] ~ Grazing[g2], method = "bray")
# F = 1.07; P = 0.371
```

```
g2 <- Grazing %in% c("No_No", "Yes_Yes")
set.seed(42)
adonis2(Oak1[g2, ] ~ Grazing[g2], method = "bray")
# F = 3.18; P = 0.001
```

Putting this together in a table of P-values:

	No_No	Yes_No
Yes_No	0.371	
Yes_Yes	0.001	0.943

What we did here – subsetting the data – is equivalent to what was done in `pairwise.adonis2()` in the 'Complex Models' chapter. However, these differences do not necessarily mean that one of these approaches is better than the other – there is some debate among statisticians about which approach is preferred.

Also, the more detailed approach that I outline in this appendix has some advantages:

- It draws on all of the data and hence has a more accurate estimate of the within-group variability.
- It is necessary if you want to make other planned comparisons such as comparing two levels of a factor against another level of that factor (see 'Custom Contrasts' below).
- Individual contrasts can be easily included in models that also include other terms.

Custom Contrasts

Pairwise contrasts are by no means the only contrasts that are possible. Other contrasts may be suggested by the objectives of your study. For example, you may want to evaluate whether groups of levels differ from one another.

In the context of this example, a reasonable contrast would be to compare stands that were grazed

historically ('Yes_No' and 'Yes_Yes') with those that were not ('No_No'). To do so, we follow the same procedure outlined above for designating our own pairwise contrasts.

1. Create a duplicate of the original factor.
`Grazing.custom <- Grazing`
2. Designate our own contrasts for this term. Here, we want to compare historically grazed and ungrazed stands, so our desired contrast is (1, -1, -1). No coefficients are 0 as we are not excluding any levels from this contrast. Apply this set of contrasts to the model matrix.
`Grazing.custom.mm <- model.matrix(~ C(Grazing.custom, c(1, -1, -1)))`
3. Use indexing to create a separate object for each contrast.
`con.NNvYN_YY <- Grazing.custom.mm[, 2]`
`con.NNvYN_YY.rest <- Grazing.custom.mm[, 3]`
As before, the last contrast here is one that we may not be interested in but need to include for completeness.
4. Replace the grazing term in `adonis2()` with our new contrasts, with the 'focal' contrast as the first one in the model.
`set.seed(42)`
`adonis2(Oak1 ~ con.NNvYN_YY + con.NNvYN_YY.rest, method = "bray")`

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ con.NNvYN_YY + con.NNvYN_YY.rest, method = "bray")
      Df SumOfSqs      R2      F Pr(>F)
con.NNvYN_YY      1    0.7693 0.06635 3.1677 0.001 ***
con.NNvYN_YY.rest  1    0.1402 0.01209 0.5773 0.943
Residual          44   10.6854 0.92156
Total             46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again, we have fully partitioned the df and variance associated with the grazing term into that associated with our contrast of interest and that associated with other levels of the term. This contrast indicates that past grazing status has a significant effect on composition.

You may notice that the question that we asked here is identical to asking about the past grazing status as defined in the oak plant community dataset. Indeed, what we did here is identical to using those factors in the PERMANOVA:

```
set.seed(42)
adonis2(Oak1 ~ GrazPast + GrazCurr, data = Oak, method = "bray")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = Oak1 ~ GrazPast + GrazCurr, data = Oak, method = "bray")
```

```

      Df SumOfSqs      R2      F Pr(>F)
GrazPast  1    0.7693 0.06635 3.1677 0.001 ***
GrazCurr  1    0.1402 0.01209 0.5773 0.943
Residual 44   10.6854 0.92156
Total    46   11.5949 1.00000
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Our custom model matrix produces exactly the same output as when we analyze GrazPast and GrazCurr. **This is not unique to PERMANOVA – is part of the logic behind all linear models in R.**

Verify that, as with the other contrasts above, the order of these terms can dramatically affect the conclusions. Again, **this is not unique to PERMANOVA – it is a consequence of model structure and the fact that these data are unbalanced.** The same issues apply to any linear model in R.

References

Crawley, M.J. 2012. *The R book*. Second edition (First edition, 2007). Wiley, West Sussex, England.