Ordination and Analysis of Dissimilarities: Tutorial with R and vegan

Jari Oksanen

July 24, 2009

Contents

1 Introduction

This tutorial leads you some typical paths of numerical analysis of community data using R package **vegan**. The tutorial shows the commands and explains some results, but usually you must repeat the commands to see the output. The R commands and design principles are explained under way dispersed in the text of multivariate analysis. If you are familiar with R, you can skip those parts.

The input text is shown in slanted typewriter font. Each line of input is preceeded by a prompt > followed by the command. You should not type the prompt, but only the text following the prompt. In some cases the command may be so long that it continues to the next row. In that case the continuous prompt is + which should not be written. Possible output is shown as upright typewriter font. An example will look something like this:

```
> This_is_a_line_with_an_input_command(and_a_long_line,
+  continuing_here)

Here is a line of ouput
```

Sometimes you can see a continuation prompt even when you do not expect one. Normally this happens because you have opened a structure that is not closed. Perhaps you had opened a quote with ", but you forgot to have the closing quote. In that case you can bail out using ctrl-c key combination.

1.1 Getting In Your Data

This text uses standard **vegan** data sets which are directly available for any user. You can run this tutorial with your own data, but first you have got to read your data into R. If you do not want to use your own data, you can skip this section.

The **vegan** package uses convention were community data and environmental data are kept in separate data frames. Typically we need to read in two data sets. The rows must match between data sets.

R can read in several types of data, for instance from SPSS, SAS or Stata using standard pakcage **foreign**. R ships with a special pdf manual on data input and output. You should consult that manual for efficient data input. In addition, **vegan** has function `read.cep` that can read in CANOCO formatted data sets popular among ecologists. You should consult manuals with those data sets. In this tutorial I only briefly explain how to handle data input from spead sheet data files.

Perhaps the easiest way is to save the spread sheet file as comma separated values (csv) in plain ascii. Such an option is available in all spread sheet programs. Further, **vegan** – and probably all other packages – assume that rows

are sampling units and columns are variables or species. R assumes that both rows and columns have names. So you should prepare your spreadsheet so that its first row contains column names and first column contains row names. The column names are more important, and you can well continue with unnamed rows. Then you save your file as a csv. In ideal case, a miniature csv file looks like this:

```
,Belper,Empnig,Junbuf,Junart,Airpra,Elepal,Rumace,Viclat
plot2,3,0,0,0,0,0,0,0
plot13,0,0,3,0,0,0,0,0
plot4,2,0,0,0,0,0,0,0
plot16,0,0,0,3,0,8,0,0
```

This file is read into R using command `read.csv` (or in many non-English localities with `read.csv2`):

```
> myfile <- read.csv(file.choose(), head = TRUE, row.names = 1)
```

You can give the name of your file as the arguments, but if you do not name the complete path to your file (like is typical in Windows), you can launch a command `file.choose` that opens a file browser. The argument `head` tells that you do have column names, and the argument of `row.names` gives the column where row names are found (and alternatively, it can be something else, but see `?read.csv` for details). You must save your data into a data frame with the name of you choice or the read data will be dumped into your screen and disappear.

Be careful with names. You must type variable names or plot species names and it is good to keep them short to be easily typed but long enough to be easily interpreted afterwards. Do not use spaces in names and avoid special characters.

After reading in the data, you must check that everything is correct. You can have a look at the general `str`ucture of your data with:

```
> str(myfile)
```

For community data, all variables should be numeric and there should be no missing values. You can also ask for a `summary`:

```
> summary(myfile)
```

If you had blanks instead of zeros for some entries, these are normally interpreted as missing values and marked as `NA`. You can change all `NA` into zeros with the following:

```
> myfile[is.na(myfile)] <- 0
```

If you read in environmental data, you should check your data types. Continuos variables should be numeric and class variables should be factors. If you coded your class variables with characters, they will be interpreted as factors

automatically. It is very common to code factor levels as numeric in spread sheets. This should be usually avoided in R because character labels are easier to intepret and will be automatically made into factors. If you data set has a factor called `Factor` which was wrongly interpreted as numeric, you can change it into factor as:

```
> myenvfile$Factor <- factor(myenvfile$Factor)
```

You should never code your factors as "dummy" variables. R does that much better and treats factors correctly.

The basic data object is called data frame in R. It may look deceivingly like a matrix, but it is deeply different. Matrix can only hold one type of data, typically numbers, but data frames can hold variables of different types. It is often impossible to see whether you have a data frame or a matrix, but the difference can cause errors. You can ask `class(myfile)` to see which one you have. Sometimes the data types are changed surprisingly. If you have read your data in transposed form, that is, species as rows and sampling units as rows, you can transpose the file to the normal with command `t()`. However, this also changes your data into a matrix which may surprise you later. So you should do like this:

```
> mynewfile <- as.data.frame(t(myfile))
```

It is also wise to have different names for the new and old fail: recovery from errors is much easier when you have not destroyed your original data.

2 Simple Ordination

2.1 Nonmetric Multidimensional Scaling (NMDS)

First we need to load library **vegan** and take into use some data sets available in this package:

```
> library(vegan)
> data(dune)
```

Running NMDS takes only one command:

```
> m <- metaMDS(dune)
```

Function `metaMDS` is a from **vegan** library. It calls internally function `isoMDS` of the **MASS** package for the proper ordination work. Function `metaMDS` provides a simple wrapper to a one-shot NMDS ordination following the recommended procedures. The steps taken are described in the help page of `metaMDS` which you can read with command

```
> help(metaMDS)
```

Read the help page and explain what are the steps takein in the command.

Sometimes the `metaMDS` does not find the same best solution twice, and in that case you cannot be sure that the found best solution really is the global optimum. If you are uncertain, you can continue the iterations from your current solution by giving the name of your solution in the argument `previous.best`:

```
> m <- metaMDS(dune, previous = m)
```

We saved the results of the analysis in object called `m`. You can use any decent name. Decent names do not have spaces, they do not begin with a number, and they do not contain special characters that can be interpreted as mathematical operators such as `+-/:^`. If you do not save the results, they will be printed on the screen and will be lost once the screen scrolls out from the sight. The symbol `<-` saves the results, but you can also use more familiar `=`. If you type the name of your result object, the object will be printed on the screen:

```
> m
```

However, this is only a printed presentation of the object, and the the result object containts much more detailed information. For instance, the ordination scores are not displayed, although they also are saved. You can see this by plotting the object:

```
> plot(m)
```

The default plot command used black circles for sample plots and red crosses for species. You can see the names for both by defining argument `type = "t"` for text:

```
> plot(m, type = "t")
```

The `metaMDS` function decided many things for you. For instance, you did not need to define or calculate the dissimilarity index, but the function automatically used Bray-Curtis which is a popular choice. However, you can use any of the indices defined in function `vegdist`, or you can even define the name of some other function to calculate the dissimilarities. For instance, you can use the Euclidean distances which are commonly regarded as a poor choice for community data:

```
> m2 <- metaMDS(dune, dist = "euclid")
> m2
```

You can compare the shape of the nonlinear regression with these two method using function `stressplot`:

```
> stressplot(m)
> stressplot(m2)
```

How do the stress plots differ from each other?

You can directly compare the similarity of the results using Procrustes rotation (do so, and describe the differences):

```
> plot(procrustes(m, m2))
```

2.2 Eigenvector Ordination

R has several functions for running Principal Components Analysis (PCA), and many packages implementing variants of Correspondence Analysis (CA). Here we only show how to use **vegan** functions `rda` and `cca` for these tasks.

2.2.1 Principal Components Analysis (PCA)

PCA can be run with command `rda`, or you can use standard R commands `prcomp` or `princomp`. We shall later use `rda` for constrained ordination, and therefore we use it also for PCA:

```
> ord <- rda(dune)
> ord
```

The results can be plotted with many alternative scaling systems for sites and species. The scales are described in `help(scores.cca)`, and discussed deeply in vignette on *Design Decisions* which can be accessed using **vegan** command `vegandocs()`. You can inspect the visual effects by scrolling through the alternatives:

```
> plot(ord)
> plot(ord, scal = 1)
> plot(ord, scal = 3)
> plot(ord, scal = -1)
> plot(ord, scal = -2)
> plot(ord, scal = 2)
```

Study these alternative plot scalings. In particular, you should understand what happens with negative scaling values.

In addition to standard `plot` function, you can also use `biplot` function which uses arrows for species instead of points. Make clear to yourself why arrows are used (see lectures). How should the expected abundances of species be interpreted with arrows with different scales?

```
> biplot(ord, scal = 2)
> biplot(ord, scal = -2)
```

Negative scalings display species as scaled (what does this mean? – check in lectures), although the species are not analysed as scaled. To give equal weights to all species, you should specify a new argument in the call:

```
> sord <- rda(dune, scal = TRUE)
> sord
> plot(sord)
> biplot(sord)
```

How the results differ from unscaled PCA?

Eigenvector ordinations implement linear mapping of Euclidean distances onto ordination (check lectures). Nonmetric Multidimensional Scaling (NMDS)

used nonlinear mapping of any distance or dissimlarity measure. You can study the effect of both nonlinear mapping and non-Euclidean distances using Procrustes rotation:

```
> plot(procrustes(m, ord))
> plot(procrustes(m2, ord))
```

2.2.2 Correspondence Analysis (CA)

CA analysis is simlar to PCA, but uses command `cca` instead of `rda`:

```
> mca = cca(dune)
> mca
```

Compare the output of `cca` to that of `rda` and find out how the differ (lecture slides may be useful).

The plotting happens similarly as in PCA, and again there are several scaling alternatives; their description can be found with `help(scores.cca)`, and inspected visually:

```
> plot(mca)
> plot(mca, scal = 1)
> plot(mca, scal = 2)
> plot(mca, scal = 3)
```

What are the main differences among these scores? This is mainly related to the process of weighted averages (discussed in lectures).

3 The Anatomy of an R plot

We have already used standard `plot` function for ordination result. This is a quick and dirty solution to give the first impression on the results. For clean or publishable results we may need to exercise a closer control on the results, and therefore we need to understand the inner workings of R `plot`s.

The basic `plot` command does not only draw a graph, but most importantly it sets the plotting area and plotting scales. After `plot` you can add several new elements to the created plot. For instance, you add texts in margins, and therefore the `plot` command reserves empty space in the margins:

```
> plot(mca)
> title(main = "Correspondence Analysis")
```

There is empty space in the margin in case you want to add some comments there. Often you do not want to have anything there, and you can set narrower margins to get a larger plotting area. There are many other graphical parameters that can be set. Many of these are described in `help(par)`. The margins are set by `par`ameter called `mar` which sets the margins as four numbers of margin widths in lines (rows of text) in order bottom, left, top, right. The following sets a bit narrower margins than the default:

```
> par(mar = c(4, 4, 1, 1) + 0.1)
> plot(mca)
```

You have only a limited control of basic `plot`. For instance, you can select either `type = "t"` (text) or `type = "p"` (points), but you cannot use points for plots and text for species, or you cannot select colours or sizes of symbols separately. However, you can first draw an empty plot (`type = "n"`, none), and then use commands `points` or `text` that *add* items to an existing plot:

```
> plot(mca, type = "n")
> points(mca, display = "sites", col = "blue", pch = 16)
> text(mca, col = "red", dis = "sp")
```

Both functions are configurable as you see in `help(text)` and `help(points)`. Plotting characters (`pch`) of `points` can be given either as numbers (described in the help page), or as symbols (such as `pch = "+"`). For a quick survey of choices you can use commands `example(text)` and `example(points)`.

3.1 Congested Plots

Ordination plots are often crowded and messy: names are written over each other and may be different to read. For publications you need cleaner plots.

One alternative is to use opaque labels for text with function `ordilabel`. These will still cover each other, but at least the uppermost will be readable. With argument `priority` you can select which cases are uppermost. The following draws an empty plot, adds sample plots as points, and then species names on opaque labels giving higher priority to more abundant species (high column sums):

```
> plot(mca, type = "n")
> points(mca, display = "sites")
> abu <- colSums(dune)
> ordilabel(mca, col = "red", dis = "sp", fill = "peachpuff", priority = abu)
```

Another alternative is to use function `orditorp` which uses text only if this does not cover previously added names, and points otherwise. The function also knows the `priority` argument:

```
> plot(mca, type = "n")
> points(mca, display = "sites", pch = 23, col = "red", bg = "yellow")
> orditorp(mca, dis = "sp", prio = abu, pch = "+", pcol = "gray")
```

Finally there is function `ordipointlabel` which uses both points and text to label the points. The function tries to locate the text to minimize the overlap. This is a slow numerical process, and will reach different results in most times, but can sometimes give decent results automatically (there are data sets with so many species or plots that it is impossible to label all neatly).

```
> ordipointlabel(mca)
```

For a complete control of created plot you can use interactive command `orditkplot` which also uses points and labels for the plots. The points are in fixed positions, but their labels can be moved with mouse. The edited plots can be saved in various graphical file formats or dumped back to the R session for further manipulation and plotting with `plot` command. In its basic form, the function only accepts one kind of scores, and the default is to plot species:

```
> orditkplot(mca)
```

Edit this plot and save it as a pdf file for a manucript, or as jpeg file for a web page.

However, you can save the invisible result of `ordipointlabel` and further edit the result. In this case you will automatically get both species and sites into same editable graph with different plotting symbols and text colours:

```
> pl <- ordipointlabel(mca)
> orditkplot(pl)
```

3.2 Alternative Plotting Functions

In addition to standard `plot` and its associates (`points`, `text`, etc.), there are some alternative plotting functions. One useful function for inspecting results is `ordirgl` that allows spinning of 3D graphics:

```
> ordirgl(mca, size = 3)
```

You can spin this graph around by pressing down left mouse button, or zoom into plot using right button (the buttons will be different in Mac). You can also add "spider plots" which connect sites to centroids of classes. The following will connect each site with certain Management practice to its centroid, and spinning this plot will help to see how these treatments differ in 3D:

```
> data(dune.env)
> attach(dune.env)
> orglspider(mca, Management)
```

Finally, you can add species names to the grap:

```
> orgltext(mca, dis = "sp", col = "yellow")
```

As a second example we examine a larger data set where a number of Dutch ditches were poisoned with insecticide Pyrifos, and the effects of the impact and recovery of animal plankton was followed. The sampling design is regular and the environmental data are automatically generated with command `example`, and the data set is described more thoroughly in `help(pyrifos)`:

```
> data(pyrifos)
> example(pyrifos)
```

We use Detrended Correspondence Analysis (DCA) to demonstrate also that method:

```
> dca <- decorana(pyrifos)
> dca
```

Compare the display of this analysis to other methods. The following gives ordinary CA for comparison (but does not save the results):

```
> cap <- cca(pyrifos)
> cap
```

Compare the eigenvalues. Why should they differ? (Lectures may help here.) Compare the results visually using Procrustes analysis.

```
> plot(procrustes(cap, dca))
> plot(procrustes(dca, cap, choices = 1:2))
```

The default number of extracted axes differs between `decorana` and `cca` results, and therefor the first command rotates four DCA axes to two CA axes (and projections of these four axes are shown in the graph). The latter command explicitly chooses axes 1 to 2.

We can see if there is a visual sign of DCA artefacts ("lollypaper", "lasagna" effects) with 3D dynamic plot, where we also use different colours for Pyrifos doses:

```
> ordirgl(dca, size = 3, col = as.numeric(dose))
```

Spin this plot to get the shape of the cloud of points, and to see how the dose influences the pattern. For a clearer pattern, we add lines connecting consecutive observations within ditches:

```
> orglsegments(dca, ditch)
```

Spinning is good for private use since it helps in understanding 3D structures, but it is difficult for papers. An alternative plotting command used Lattice or Trellis commands to produce panels of plots or alternative special plots. The following produces separate panels for each level of Pyrifos, displays each ditch with a different colour, and connects points within ditches by lines showing the temporal succession.

```
> ordixyplot(dca, form = DCA2 ~ DCA1 | dose, group = ditch, type = "b")
```

The `form` says that draw axis DCA2 against DCA1 making panel for each `dose`, and use lines and points (`type = "b"` or "both" lines and points) for each `ditch`. Please note that the sign before `dose` is not a slash but a vertical bar (`|`). Explain with this graph how the natural annual succession and the impact of Pyrifos can be seen. You can identify the species for both variables by looking at the ordinary plot of species scores:

```
> plot(dca, dis = "sp")
```

If this is crowded, you can use tricks for congested plots to produce a more readable version.

4 Fitting Environmental Variables

The basic command to fit vectors or factor levels of environmental variables is `envfit`. The following example uses the two alternative result of NMDS:

```
> data(dune.env)
> envfit(m, dune.env)
> ef <- envfit(m, dune.env, perm = 1000)
> ef
> ef2 <- envfit(m2, dune.env, perm = 1000)
> ef2
```

Which environmental variables are most important? Which of these alternative ordinations seems to be better related to the environment?

We can add the fitted variables to the model as vectors and centroids of factor levels. The vectors will be automatically scaled for the graph area. The following shows only the variables that were regarded as statistically significant at level $P \leq 0.05$:

```
> plot(m)
> plot(ef, add = T, p. = 0.05)
> plot(m2)
> plot(ef2, add = T, p. = 0.05)
```

The basic command fits vectors and factors for all variables in the environmental data frame. If we use formula interface, we can select only some of the variables from the `data`:

```
> plot(m)
> plot(envfit(m ~ Management, data = dune.env), add = TRUE)
```

4.1 Display of Factors

The plots of factor fitting will only show the class centroids. We may be interested in seeing the variation or scatter of class members. Some commands add graphical descriptions of the items into an existing plot:

```
> attach(dune.env)
> ordispider(m, Management, col = "skyblue")
> ordihull(m, Management, col = "pink")
> ordiellipse(m, Management)
> ordiellipse(m, Management, kind = "se", conf = 0.95, col = "red")
```

Function `ordispider` connects class members to their centroid with lines, `ordihull` draws a convex hull enclosing all points, and `ordiellipse`[1] draws (in this case) 95 % confidence ellipses around class centroids. If these confidence ellipses do no overlap, the classes probably are significantly different at level $P \leq 0.05$.

[1] Function requires package **ellipse** which may not be installed in your computer. You can install it locally using command `install.packages("ellipse", lib.loc="f:/myusbstick")`, where the `lib.loc` argument gives an address of a directory ("folder") that you can write to.

4.2 Fitting Surfaces

Vector fitting implies a linear trend surface. Check in lecture slides what this mean, and how vectors should be interpreted. In the following graph, you can estimate the the relative thickness of A1 horizon in different plots:

```
> data(dune.env)
> plot(m, dis = "sites", type = "t")
> ef <- envfit(m ~ A1, dune.env)
> plot(ef, add = TRUE)
> attach(dune.env)
> sf <- ordisurf(m, A1, add = TRUE)
```

Function `ordisurf` fits a flexible surface, and it automatically estimates how linear or nonlinear the trend surface will be. The function uses thinplate splines with generalized crossvalidation to assess the number of degrees of freedom for these surfaces. If the linear trend surface implied by the model is adequate, the isocline values of the trend surface will be equally spaced lines perpendicular to the fitted vector. Sometimes this is true, but not always.

5 Constrained Ordination

The following constrained ordination methods are available in **vegan**:

1. `rda` for redundancy analysis (RDA), based on principal components analysis (PCA)
2. `cca` for constrained correspondence analysis (CCA), a.k.a. canonical correspondence analysis, and based on correspondence analysis
3. `capscale` for distance-based redundancy analysis (db-RDA), based on metric multidimensional scaling, a.k.a. principal coordinates analysis (PCoA).

These three functions work similarly, and have a similar user interface. You can freely select your favourite, although this tutorial focuses on CCA with some sidetracks to RDA.

The following data sets in **vegan** have both community data and environmental data, and can be used in constrained analysis, although only the Dutch dune meadow data and East Fennoscandian reindeer pastures are used in this tutorial:

- Reindeer pastures with 24 sites and 44 species (`varespec`), and environmental data with 14 continuous soil variables (`varechem`).
- Dutch dune meadows with 20 sites and 30 species (`dune`), and environmental data with four factors (some of these ordered) and one continuous variable (`dune.env`).

- Oribatid mites with 70 soil cores and 35 species sampled in a 2.5×10m plot (`mite`). The environmental data contains two continuous and three factor variables (`mite.env`). In addition, there are data sets of spatial coordinates of cores (`mite.xy`) and principal coordinates of neighbourhood matrix (PCNM) derived from these (`mite.pcnm`).
- Planktic animals in 12 mesocosms sampled 11 times after Pyrifos treatment. The community data has 132 observations and 178 species (`pyrifos`). The environmental data on experimental design can be generated using command `example(pyrifos)`.

5.1 Preliminaries: Inspecting Data

We study first the lichen pasture data with only continuous constraints. The data sets are taken into use with:

```
> library(vegan)
> data(varespec)
> data(varechem)
```

The environmental data can be inspected using commands `str` which shows the `str`ucture of any object in a compact form, and asking for a `summary` of a data frame:

```
> str(varechem)
> summary(varechem)
```

There is a difference between a data frame and a matrix in R: data frame is actually a list of variables which can be of different types (continuous, factors, ordered factors), whereas matrix only contains numbers of the same type.

The dependencies among variables can be inspected visually using `plot` command for data frames; under the hood this calls `pairs` function

```
> plot(varechem, gap = 0, panel = panel.smooth)
```

It always useful to have a look at the data before rushing into analysis. It is necessary to have a look at the data if you read in your own data: you really must check that the data were imported correctly.

You can refer to rows and columns in the data frame in various ways. One way is to use numeric indices within square brackets `[]`. The first item refers to a row, the second item to a column:

```
> varechem[3, 7]
```

Several items can be referred to by combining indices within `c()` or by using a regular sequence with `:` between first and last item:

```
> varechem[2, c(3, 1, 7)]
> varechem[3:5, 7]
```

If you omit one index, whole row or whole column will be given to you:

```
> varechem[2, ]
> varechem[, 7]
```

Finally, you can also refer to rows and columns by their names instead of numeric indices, and in data frames you can also use the dollar sign for variables:

```
> varechem[, "pH"]
> varechem$pH
```

5.2 Simple Case of Constrained Ordination

If you only have continuous variables, you can constrain the ordination by adding a second argument to the call of `cca` or `rda`:

```
> m <- cca(varespec)
> mm <- cca(varespec, varechem)
> m
> mm
```

Compare the output of these two ordinations. In particular, see how the inertia and and rank (number of axes) are decomposed in constrained ordination. The only way to select environmental variables from the full data set is to use a subset of the matrix:

```
> cca(varespec, varechem[, c("Al", "P", "K")])
```

For a full numerical survey, you can use `summary` command[2]

```
> summary(mm)
```

The `summary` also shows how the total inertia is divided between individual axes, and the lines for accounted inertia shows the accumulated variation explained by all axes, as well as the accounted constrained inertia. Note also that scaling of the site and species scores happens only when you look at them in `summary`, and you can give `scaling` as an argument to the summary. Note also that there are two different kind of site scores in the output. The lectures explain their difference.

The two kind of site scores are often called as WA scores and LC scores. The WA scores are derived from species scores, and the LC scores are derived from constraints as their linear combinations. The analysis tries to keep these two sets as similar as possible. Their similarity can be inspected with species–environment correlation', which simply is the correlation between LC and WA score for an axis.

```
> spenvcor(mm)
```

[2]From **vegan** version 1.15-1 you can use `head(summary(mm))` to show only some lines of the scores, but in older versions your screen will be filled with numbers, and you msut scroll back to see the output.

The default plot displays species scores, WA scores and environmental variables.

```
> plot(mm)
```

You can change this by givint the displayed elements in argument `display`. The following shows only LC scores and environmental variables:

```
> plot(mm, display = c("lc", "bp"))
```

The items are called `"sp", "wa", "lc", "bp", "cn"`, which are self explanatory, except the last which refers to the centroids of the environmental variables. You can visually inspect the difference between WA and LC scores (or the species – environment relationship) with command `ordispider` which (among other alternatives) joins WA and LC scores by a line:

```
> plot(mm, dis = c("wa", "lc"))
> ordispider(mm)
```

It is often a bad idea to use constrained ordination if you have a very large number of constraints: probably you will not constrain at all, but your analysis may be very similar to ordinary unconstrained analysis that could have been used just as well. You can inspect this with Procrustes rotation:

```
> plot(procrustes(m, mm))
```

You can also see how similar the ordinations are by fitting environmental variables to unconstrained ordination.

```
> plot(m)
> plot(envfit(m, varechem))
```

5.3 Formula Interface

You can also use formula interface to select the variables used as constraints:

```
> cca(varespec ~ Al + P + K, data = varechem)
```

With formula interface you can also use factor constraints:

```
> data(dune)
> data(dune.env)
> str(dune.env)
> summary(dune.env)
> mdun <- cca(dune ~ Management + A1, dune.env)
> mdun
> plot(mdun)
```

Note here how the use of factors increases the rank of the constrained solution. If you have n classes, you will get $n-1$ axes — provided your factor levels really are independent.

5.4 Model Choice

With formula we have a full control of the model, but we face with the problem of model choice. Models must be built carefully, and preferably used to test specific hypotheses. Sometimes we may want to use automatic model building, but this must be done carefully. There are some shortcuts and tricks for this in **vegan**, but these should be used with utmost care.

In automatic model building we usually need two extreme models: the smallest and the largest model considered. The following shortcuts build a model with all environmental variables and a model with no environmental variables, but both with a formula so that terms can be added or removed from the model:

```
> m1 <- cca(varespec ~ ., varechem)
> m0 <- cca(varespec ~ 1, varechem)
> m1
> m0
```

Then we can use `step` function to select the "best" model. The `step` function uses Akaike's Informatin Criterion (AIC) in model choice. The AIC is based on the goodness of fit (high constrained inertia), but it is penalized by the number of estimated parameters (constrained rank). The alternative models are ordered by AIC. In each case, `+` indicated the effect of adding a term, and `-` the effect of removing a term, while the current model is marked as `<none>`. The model building proceeds by steps until the current model (`<none>`) is the best. Special care is needed, because there really is no AIC for constrained ordination (although it is calculated!), and we always should inspect the validity of model choice. One way is to ask for approximate significance tests. If the model choice was valid, all included variables (with `-` before their name) should be significant, and all excluded variables (with `+` before their name) should be insignificant.

```
> m = step(m0, scope = formula(m1), test = "perm")
```

With continuous variables, the model building often works rather well, but it should not be trusted blindly. You can see this if you use `rda` instead of `cca`, or `dune` and `dune.env` data sets instead of lichen pastures (you may try). Moreover, you may end up with different models if you change the modelling strategy. The following simplifies the maximum model:

```
> mback <- step(m1, test = "perm")
```

Compare the order in which variables were added in the first model, and removed in this model.

One problem with model building is that constraining variables are not independent, but they are correlated. Anyone of the correlated variables can be explained with other variables. Such variables are redundant ("expendable") when they are with other variables, but they may be the best variables along and prevent other variables to enter the model. A statistic describing this is called variance inflation factor (VIF) which is 1 for completely independent variables, and values above 10 or 20 (depending on you r taste) are regarded as

higly multicollinear (dependent on others). The VIF of a variable will depend on the set it sits with:

```
> vif.cca(m1)
> vif.cca(m)
> vif.cca(mback)
```

5.5 Updating a Model

You can manually build models if the automatic procedure fails. For instance, it does not work for Dutch dunes, where the building stops too early:

```
> m0 <- cca(dune ~ 1, dune.env)
> m1 <- cca(dune ~ ., dune.env)
> m <- step(m0, scope = formula(m1), test = "p")
> m
```

You can use `update` command where you change the formula. The retained parts of the formula are shown by a dot (`.`) and terms are added with `+` or removed with `-`. The following adds `Management`:

```
> m <- update(m, . ~ . + Management)
```

Then you can see if any other terms should be added to this model, or if removing an included term could improve the model – if constraints are correlated, the significance can change with adding or removing variables from the model:

```
> add1(m, scope = formula(m1), test = "perm")
> drop1(m, test = "perm")
```

If needed, you can manually continue model building.

5.6 Significance tests

We already saw significance tests with model building. These tests were based on permutation: there is no known distribution of inertia that could be used for strict statistical testing. We simply permute the rows of community data, repeat the analysis, and get a random result. If our observed result is better than most of the random models (say, better than 95% of them), we say that our results are significant.

Package **vegan** has several alternative types of significance tests. They all can be performed with a function called `anova`. The name is somewhat misleading: the test are based on permutations although the layout of the results is similar as in the standard ANOVA table. The default is an overall test of all variables together:

```
> anova(m, by = "ma")
```

The function actually used was `anova.cca`, but you do not need to give its name in full, because R automatically chooses the correct `anova` variant for the result of constrained ordination.

The `anova.cca` function tries to be clever and lazy: it automatically stops if the observed permutation significance probably differs from the targeted critical value (0.05 as default), but it will continue long in uncertain cases. You must set `step` and `perm.max` to same values to override this behaviour.

It is also possible to analyse terms separately:

```
> anova(m, by = "term", permu = 200)
```

In this case, the function is unable to automatically select the number of iterations. This test is sequential: the terms are analysed in the order they happen to be in the model. You can also analyse significances of marginal effects ("Type III effects"):

```
> anova(m, by = "mar")
```

Moreover, it is possible to analyse significance of each axis:

```
> anova(m, by = "axis", perm = 500)
```

Now the automatic selection works, but typically some of your axes will be very close to the critical value, and it may be useful to set a lower `perm.max` than the default 10000 (typically you use higher limits than in these examples: we used lower limits to save time when this document is automatically generated with this package).

5.7 Conditioned or partial ordination

All constrained ordination methods can have terms that are partialled out from the analysis before constraints:

```
> m <- cca(dune ~ A1 + Management + Condition(Moisture), data = dune.env)
> m
```

This partials out the effect of `Moisture` before analysing the effects of `A1` and `Management`. This also influences the signficances of the terms:

```
> anova(m, by = "term", perm = 500)
```

If we had a designed experiment, we may wish to restrict the permutations so that the observations only are permuted within levels of `strata`:

```
> with(dune.env, anova(m, by = "term", perm = 500, strata = Moisture))
```

Here `with()` is a special function that makes variables in `dune.env` visible to the following command. If you only type `Moisture` in an R prompt, you will get an error of missing variables. Functions with formulae have a `data` argument giving the name of the data frame from which the variables are found, but other

functions usually do not have such an argument. Instead of `with(dune.env, command())`, you can first `attach(dune.env)` and after that all variables in the data frame are visible in the session. This may be dangerous if you have similar names in your session and several `attach`ed data frames: it is difficult to know which of these was used.

6 Comparison of Dissimilarities and Distances

There are two competing — even hostile — approaches for comparison of geographical distances, ecological distances and community dissimilarities: Direct comparisons of distances and dissimilarities with Mantel tests and related methods, and partitioning of variance in data generating the distances. Both of these methods have their drawbacks. Mantel tests are a natural choice if your data really involve geographical distances (instead of geographical locations), and you are interestested in the effects of distances. This seems to be a popular subject recently, and people working with small things (like microbes, fungi and algae) like to think that their organisms are ubiquitous and not influenced by the geographical distances like the bigger things (lichens, some bryophytes, plants). It sounds more arbitrary to transform environmental data into environmental distances and community composition to community dissimilarity. Having more than two sets of distances or dissimilarities may be mathematically tricky, and the methods may not be very powerful. Variance partition is a natural approach with rectangular environmental and community data, but mapping geographical distances to a rectangular data may be problematic (and I do think it is more problematic than indicated in the literature).

This section analyses the classic Oribatid mite data that is available in **vegan**. The Oribatids were sampled in 2.5×10m grid. The `mite` data contains the counts of Oribatid species in soil cores, `mite.env` contains the environmental data, `mite.xy` the spatial coordinates, and `mite.pcnm` the principal coordinates of neighborhood matrix (PCNM) derived from `mite.xy`. If you use your own data sets and want to get PCNM scores, you can use packages **PCNM** or **veganSedar** in `http://sedar.r-forge.r-project.org`. We first take these data sets into use:

```
> data(mite)
> data(mite.env)
> data(mite.xy)
> data(mite.pcnm)
```

6.1 Mantel Test

The Euclidean distances between sampling locations can be found with function `dist` in base R or with `vegdist method = "euclid"`. There are numerous alternatives for community dissimilarities. Function `vegdist` in **vegan** contains some popular ones for community ecology, and many other packages have other

alternatives (**ade4** and **labdsv** are some useful alternative packages). In addition, **vegan** has function `designdist` that allows you to write your dissimilarity functions. Here we use `betadiver` in **vegan** that finds indices of beta diversity, and most of them are pairwise dissimilarity indices. We select index z of the species-area model $S = cA^z$ with nice interpretation that dissimilarity values of $z < 0.3$ are random noise of replicate samples. This is a binary index so that now we do not consider transformatin of Oribatid counts:

```
> geodis <- dist(mite.xy)
> z <- betadiver(mite, "z")
> plot(geodis, z, xlab = "Geographical Distance (m)", pch = 21,
+     bg = "white", cex = 0.8)
> abline(h = 0.3, col = "red")
> abline(lm(z ~ geodis), lwd = 2, col = "blue")
```

The scatter diagrams in R look often messy, because standard plotting character (`pch = 1`) is a ring and other points are visible through its hole (you can try). Plotting character `21` is similar, but it has an opaque background (`bg`) giving more pleasant graphs. The two `abline` commands add a horizontal line of no interest and a fit line from linear regression (`lm`).

Running Mantel test is simple:

```
> mantel(geodis, z)
```

6.2 Partial Mantel Test

Most ecologists interested in Mantel tests actually seem to like to use partial mantel test in order to separate the effects of, say, geographical and environmental distances. Let's have a look at the environmental data:

```
> str(mite.env)
```

Only two of the variables are numeric (continuous) and the others are factors or ordered factors. The ordered factor could be changed into a numeric variable, and variable `Topo` only has two levels which also can be handled as numeric, but the multiclass factor `Substrate` is more difficult. Function `daisy` in the **cluster** package can handle ordered and unordered factors without transformations with Gower distances:

```
> library(cluster)
> envdis <- daisy(mite.env)
> attr(envdis, "Types")
```

The `attr`ibutes tell that we had `I`nterval (numeric), `N`ominal (factor) and `O`rdered (ordered factor) variables. The numeric variables are automatically scaled to equal variance if the data are of mixed types.

The environmental distances influence the community composition, and they also have spatial structure:

```
> mantel(envdis, z)
> mantel(geodis, envdis)
```

Partial Mantel test is just a Mantel test using partial correlation:

```
> mantel.partial(geodis, z, envdis)
> mantel.partial(envdis, z, geodis)
```

There is no real nice way of plotting results of partial Mantel test. Although we had a linear regression in the scatter graph (and Mantel test is linear!), the residuals are not Euclidean and cannot be correctly plotted (but who cares? – except I). We could have lattice (syn. trellis) graphics where we split the distances by partialling factor:

```
> library(lattice)
> xyplot(z ~ envdis | equal.count(geodis), type = c("p", "r"),
+     cex = 0.3, lwd = 2)
```

The trellis graphics are very powerful, very configurable but not easy to use. Reading documentation, tutorials, books on R graphics and following models in examples is necessary.

6.3 Partition of Variance

Partial constrained ordination does partition variation, and can be used to similar questions of decomposing beta diversity as Mantel test. The analysis is natural with original community and environmental data, but handling geographical distances is involute. Function `varpart` in **vegan** performs partition of variance in the RDA framework. There are no canned routines for the CCA framework, but see section 6.4 for an example that also applies to CCA. The reason for having only RDA framework is that the author of the function (Pierre Legendre) thinks that adjusted R^2 should be used with partitioning of variance, and those are readily available only for RDA.

Geographical distances cannot be analysed as locations, but they are changed into PCNMs that should map neighborhood relationships at different scales onto orthogonal components. The justification of this is a bit opaque to me, and in fact, PCNMs seem to work as a very flexible spatial smoother or spatial filter. In addition to the neighborhood relationship they can model closely any environmental variables with spatial dependence.

Function `varpart` can analyse up to four different components of variance. The explanatory components can be expressed as vectors, matrices or as formula referring a single data frame. The formula allows having factors which are automatically expanded to model matrices in the function. It should be remembered that data frame and matrix are different beasts even when they look similar, and we must cast them to matrices. In the following we use the shortcut formula `~ .` where the dot refers to all variables in the data frame. Further, we use presence/absence transformation so that the results are similar as in the previous analysis:

```
> varpart(mite, as.matrix(mite.pcnm), ~., data = mite.env, transfo = "pa")
```

The results can be displayed with a `plot` command. For significance test we must fit the corresponding model as `rda`. Thus the unique spatialal variation `[a]` or `X1|X2` can be analysed as:

```
> anova(rda(decostand(mite, "pa") ~ as.matrix(mite.pcnm) + Condition(SubsDens +
+     WatrCont + Substrate + Shrub + Topo), data = mite.env))
```

For some reason the dot expansion does not work within `Condition`: I'll have a look at this and try to correct the behaviour for future versions.

6.4 Manual Partition of Dissimilarities

The canned variance partition is only available for Euclidean metric (RDA), but we can perform the analyses manually. If we do not insist on using adjusted R^2 we can partition the Chi-squared variation of `cca`. The example shows how the analysis is performed with beta diversity measure z that we used in the Mantel example. We can first have a look at the variation jointly explained by spatial and environmental data:

```
> m12 <- capscale(z ~ . + as.matrix(mite.pcnm), mite.env)
> m12
```

This shows one complication in distance-based RDA: with non-Euclidean distances the total variation is not a simple sum of conditional, constrained and residual unconstrained varition, but we must substract the negative variation due to non-Euclidean distances, or the imaginary component. If the imaginary component is very large, the decomposition hardly makes sense. It is within tolerance in this case, and we can proceed to extract the unique components:

```
> m1 <- capscale(z ~ as.matrix(mite.pcnm) + Condition(SubsDens +
+     WatrCont + Substrate + Shrub + Topo), data = mite.env)
> m2 <- capscale(z ~ . + Condition(as.matrix(mite.pcnm)), mite.env)
> m1
> m2
```

If we do not want to make calculations by hand, we need knowledge of gory internal details of the `capscale` result object. All constrained ordinations have similar result objects, and they are described in `?cca.object`. We need the the total variation of the `CCA` component. The shared component is the difference of the joined variation minus the unique variations:

```
> m12$CCA$tot.chi - m1$CCA$tot.chi - m2$CCA$tot.chi
```

We can proceed in the same way with `cca`. Models with higher numbers of components can also be analysed, but they need more careful manual work.

6.5 ANOVA-like Analyses

The **vegan** package has several functions for ANOVA-style analysis where dissimilarities are explained by factor variables: `anosim` for analysis of (dis)similarities (ANOSIM), `mrpp` for multiresponse permutation procedure (MRPP), and `adonis` for nonparametric analysis of variance using distance matrices. The `adonis` function can use both factors and continuous explanatory variables and handles several variables together, but the other two are confined to a single classification.

Both ANOSIM and MRPP compare within-class dissimilarities to between-class dissimilarities, but differ in the way they derive the statistics. In addition, ANOSIM uses ranks of dissimilarities making it similar to NMDS in spirit. There are several alternatives of weighting within-cluster dissimilarities in `mrpp`. The default is to weight within-class dissimilarities by group sizes n_k. The natural choice is to weight the with the number of dissimilarities within class or with $n_k(n_k-1)$, and with this choice the function also returns classification strength, another popular statistic.

All these functions are sensitive to differences in dispersion of points. A significant result does not necessarily mean that the location of the groups are different, but some of the groups may be more heterogeneous than others. This is a common problem with nonparametric methods, and not specific to these functions. However, `adonis` seems to be less sensitive to heterogeneity than others, and it is the recommended method.

I do not recommend using `anosim` in general, but its use is easy:

```
> m <- with(mite.env, anosim(z, Shrub))
> m
```

We used again `with` to make variables of `mite.env` visible to `anosim`. There is a `plot` method which in this case really seems to indicate that one of the groups is internally more heterogeneous than others:

```
> plot(m)
```

However, two of the groups are clearly more homogeneous than between-cluster dissimilarities and the results probably are reliable. If you use `anosim`, you should inspect the results graphically to see that they are trustworthy. The niche of ANOSIM is with NMDS, since ANOSIM uses ranks of dissimilarities instead of original dissimilarities.

Another non-recommended method is `mrpp` which also is easy to use:

```
> with(mite.env, mrpp(z, Shrub, weight = 3))
```

The example used `weight.type = 3`, and with this weight type the function also returns classification strength. Classification strength is a different parametrization of the MRPP statistic with this weight type, and it has the same significance.

Function `mrpp` has a sister function called `meandist` that collects the statistics on within group dissimilarities and dissimilarities between each group:

```
> m <- with(mite.env, meandist(z, Shrub))
> m
```

The `summary` of this will give MRPP statistics A with all alternative weight types and the classification strength, but for significance tests you must use `mrpp`:

```
> summary(m)
```

The result of `meandist` is actually an averaged dissimilarity matrix where the diagonal gives the averages within classes, and other elements the averages between each class. There is a `plot` method for this that displays a dendrogram of mean dissimilarities. The terminal leaves do not go to the base level, but they are at the average within class dissimilarity level. This also means that there can be reversals: a combined class is more homogeneous than some of its subclasses:

```
> plot(m)
```

The recommended method is `adonis` which implements a multivariate non-parametric ANOVA of dissimilarities. The method is also closely related to the AMOVA of geneticians. Function `adonis` does not directly study the dissimilarities but it maps the dissimilarity matrix onto principal coordinates and bases analysis on those. With Euclidean distances the mapping is exact and the method is identical to ANOVA. However, the method is more useful with other dissimilarity types.

The `adonis` function can be normally used instead of `mrpp`:

```
> adonis(z ~ Shrub, data = mite.env)
```

To replace `anosim` we could use `rank(z)` as dependent variable. All significance tests are based on permutation.

Function has formula interface, and it can also handle more complicated models:

```
> adonis(z ~ ., data = mite.env)
```

The tests are sequential: the terms are evaluated in the order as they appear in the formula. This is statistically very correct and the standard in R, but many users do not like this. They would instead prefer "Type III" tests. They are not implemented in **vegan** and we have no plans of implementing them. "Type III" test means that every term is evaluated after all other terms. This means fitting several `adonis` models, each term in turn as the last, and taking the statistics from the last term. This can be done manually, we are no planning a canned routine for this.

In the following we utilize the sequential tests to see how much spatial PCNM explain of the data after taking into account the environmental variables (and vice versa). Alone they are really good:

```
> adonis(z ~ as.matrix(mite.pcnm))
```

We had to change data frame `mite.pcnm` to get an overall analysis. This would have been an error:

```
> adonis(z ~ mite.pcnm)
```

This would analyse each PCNM component separately:

```
> adonis(z ~ ., data = mite.pcnm)
```

Now the analysis of PCNMs after removing the environmental variables:

```
> adonis(z ~ . + as.matrix(mite.pcnm), data = mite.env)
```

If you want to analyse environmental variables as a single collective matrix term, you must use something like

```
> mm <- model.matrix(~., data = mite.env)
> adonis(z ~ mm + as.matrix(mite.pcnm))
> adonis(z ~ as.matrix(mite.pcnm) + mm)
```

However, this may be too advanced for this tutorial...

6.6 Multivariate Dispersion and beta Diversity

Instead of group differences, we may be interested in within-group heterogeneity, commonly known as beta diversity (but there are other definitions, too). Function `betadisper` implements a procedure which is similar to Levene's test of homogeneity of variances. Both this test and `adonis` are based on Marti Anderson's work. Similarly as in `adonis`, the dissimilarities are first mapped onto principal coordinates, and these two functions make a fitting pair: `adonis` for differences in means, `betadisper` for differences in dispersion.

```
> m <- with(mite.env, betadisper(z, Shrub))
> m
```

Significances can be tested either with parametric ANOVA or with permutations:

```
> anova(m)
> permutest(m)
```

Moreover, pairwise contrasts between classes can be analysed using parametric Tukey's HSD:

```
> TukeyHSD(m)
> plot(TukeyHSD(m))
```

The `plot` method maps distances to class centroids in principal coordinate analysis:

```
> plot(m)
```