Unconstrained Ordination: Tutorial with R and vegan

Jari Oksanen

January 15, 2015

Contents

1 Introduction

This tutorial leads you through some typical paths of numerical analysis of community data using `R` package **vegan**. This documents shows the commands and explains some results, but usually you must type in the the commands in `R` to see the output.

The input text is shown in `typewriter font`. Each line of input is preceded by a prompt "`R>` " 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 continuation prompt is "`+` " which should **not** be written. Possible output is also shown in `typewriter font`. An example will look something like this:

```
R> This_is_a_line_with_an_input_command(and_a_long_line,
+  continuing_here)
```

```
Here is a line of output
```

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.

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:

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

Running NMDS takes only one command:

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

Function `metaMDS` is in the **vegan** library. The function calls internally a function that runs the actual NMDS. The default NMDS engine is function `monoMDS` in the **vegan** package. Function `metaMDS` provides a simple wrapper to a one-shot NMDS ordination following the recommended procedures (like explained in the lecture slides). The steps taken are described in the help page of `metaMDS` which you can read with command `?metaMDS` or `help(metaMDS)`.

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`:

```
R> 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:

```
R> m
```

However, this is only a printed presentation of the object, and the the result object contains 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:

```
R> 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:

```
R> 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:

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

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

```
R> stressplot(m)
R> 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):

```
R> 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:

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

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

Study these alternative plot scalings. In particular, you should understand what happens with negative scaling values: see `?scores.cca` for explanation.

We jumped over scaling of scores in lectures, and therefore I explain briefly here the alternatives and the theory behind these arguments. PCA can approximate the data as $x = f(\lambda, u, v)$ where u are the sample scores, v are the species scores, and λ is the eigenvalue of the axis, and $f(\cdot)$ is a function left unspecified in this tutorial. If we have all axes, then x is reproduced exactly, and for lower number of axes we have the best least squares approximation with that number of dimensions (rank). The scores u and v are unscaled, but λ has the information on the importance of axes and determines the length of each axis. With `scaling=1`, we use $x = f(\lambda u, v)$ or we scale the site scores by eigenvalues so that important axes are longer than minor axes. The species scores will be unscaled, though. With `scaling=2` we use $x = f(u, \lambda v)$ leaving sites unscaled, and with `scaling=3` we have a compromise $s = f(\lambda^{\frac{1}{2}} u, \lambda^{\frac{1}{2}} v)$ so that both species and scores are partly scaled. The species scores (or corresponding biplot arrows) are proportional to absolute change in species abundance, and abundant species typically have higher scores than minor species. The negative `scaling` values will standardize species scores so that they reflect proportional changes. The plots are often more legible, and many people prefer this scaling. However, PCA was not based on those proportional changes, and therefore this plot is misleading. If you want to have species scores with proportional changes, you should have PCA with similar data, and set `scale = TRUE` in `rda()` command (see below).

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?

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

Negative scalings display species as scaled to unit sd, although the species are not analysed as scaled. To give equal weights to all species, you should specify a new argument in the call:

```
R> sord <- rda(dune, scal=TRUE)
R> sord
R> plot(sord)
R> 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 dissimilarity measure. You can study the effect of both nonlinear mapping and non-Euclidean distances using Procrustes rotation:

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

2.2.2 Correspondence Analysis (CA)

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

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

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

The `scaling` alternatives are similar as in `rda`, and tell which set of scores will be proportional to eigenvalues, and which is left unscaled. In CA, the species scores v areaverages of site scores u with weights of community abundances x:

$$v_j = \frac{\sum_{i=1}^N x_{ij} u_i}{\sum_{i=1}^N x_{ij}} .$$

With the undefined function we can write this $v = f(u, x)$. In the final solution we have a dual relationship between scores so that their weighted averages are $v = f(u, x)$ and $u = f(v, x)$. However, the average will always be less extreme than the values from which it was calculated, and the variance of scores will decrease at each step so that u calculated from v will shrink from its original range. The degree of this shrinkage is given by the eigenvalue λ of th axis. To restore the original range, we should divide scores by the eigenvalues:

$$u = f(v, x) \qquad v = \lambda^{-1} f(u, x) \tag{1}$$

$$u = \lambda^{-1} f(v, x) \qquad v = f(u, x) \tag{2}$$

$$u = \lambda^{-\frac{1}{2}} f(v, x) \qquad v = \lambda^{-\frac{1}{2}} f(u, x) . \tag{3}$$

First case gives `scaling=1` where sites are scaled by eigenvalues, but species are unscaled. Because $\lambda \leq 1$, site scores shrink and have a smaller variance than species scores. The argument for this scaling is that it shows the importance for each gradient for sites, and it is normal that some species have their optima outside the studied range of gradients. The second alternative gives `scaling=2`, where the species really is in the middle of the sites where it occurs offerring best visual interpretation of species composition. The third choice is a compromise.

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:

```
R> plot(mca)
R> 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 `?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:

```
R> par(mar=c(4,4,1,1)+.1)
R> 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:

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

Both functions are configurable as you see in `?text` and `?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 difficult 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 labels 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):

```
R> plot(mca, type = "n")
R> points(mca, display="sites")
R> abu <- colSums(dune)
R> 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:

```
R> plot(mca, type = "n")
R> points(mca, display="sites", pch=23, col="red", bg="yellow")
R> 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 observations that it is impossible to label all neatly).

```
R> 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:

```
R> orditkplot(mca)
```

Edit this plot and save it as a pdf file for a manuscript, 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:

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

3.2 Alternative Plotting Functions

Package **vegan3d**[1] provides alternative 3D plotting functions in addition to standard `plot` and its associates (`points`, `text`, etc.). One useful function for inspecting results is `ordirgl` that allows spinning of 3D graphics:

```
R> library(vegan3d)
R> 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:

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

Finally, you can add species names to the graph:

```
R> 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)`:

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

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

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

Compare the display of this analysis to other methods. The following gives ordinary CA for comparison:

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

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

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

[1]You can install this package using the menu choices in Windows of MacOS, or command `install.packages("vegan3d")`.

The default number of extracted axes differs between `decorana` and `cca` results, and therefore 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 artifacts ("lollypaper", "lasagna" effects) with 3D dynamic plot, where we also use different colours for Pyrifos doses:

```
R> 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:

```
R> 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.

```
R> 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:

```
R> 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:

```
R> data(dune.env)
R> envfit(m, dune.env)
R> ef <- envfit(m, dune.env, perm=1000)
R> ef
R> ef2 <- envfit(m2, dune.env, perm=1000)
R> 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 $P \leq 0.05$:

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

```
R> plot(m)
R> 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:

```
R> attach(dune.env)
R> ordispider(m, Management, col="skyblue")
R> pl <- ordihull(m, Management, col="pink")
R> ordiellipse(m, Management)
R> 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` 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 $P \leq 0.05$.

We saved the result of `ordihull` above. Both `ordihull` and `ordiellipse` return an invisible return object: nothing is printed, but the result can be saved and used later. For instance, `summary` will give surface areas covered by the hulls or ellipses:

```
R> summary(pl)
```

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:

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

You can force the function to fit a linear trend surface by specifying the arguments `knots` to value `1`. You can display the observed values by using argument `bubble` that draws a circle with a diameter proportional to the observed value of the parameter. The value of `bubble` gives the maximum size of the circle:

```
R> plot(m, dis="sites", type = "n")
R> sf <- ordisurf(m, A1, add = TRUE, knots = 1, bubble = 4)
```

We saved the result returned by `ordisurf`. This is a `gam` result object (package **mgcv**), and we can use all `gam` methods with this object. In addition, **vegan** function `calibrate` can be used to find the fitted values:

```
R> calibrate(sf)
R> plot(A1, calibrate(sf))
R> abline(0,1)
```