Constrained Ordination: Tutorial with R and vegan

Jari Oksanen

January 25, 2012

Abstract

Constrained ordination methods include constrained (or canonical) correspondence analysis (CCA), redundancy analysis (RDA) and distance-based redundancy analysis (db-RDA). Constraining means that ordination only shows the community variation that can be explained by external environmental variables or constraints. This document describes the following basic concepts (1) usage of constrained ordination, (2) construction of model formulae, (3) updating of the model and choice of the constraining variables, (4) statistical significance of the overall model, single model terms or axes with permutation tests, and (5) partial ordination where the effects of some variables are removed before main ordination. The document also describes basic R concepts of inspecting data, and indexing, selecting and subsetting data.

Contents

1 Introduction

This tutorial describes the use of constrained ordination techniques in R packages **vegan**. This is a sequel to a similar tutorial describing unconstrained 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 $2.5 \times 10\,\mathrm{m}$ 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`).
- Zooplankton 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)`.

2 Preliminaries: Inspecting Data

We study first the lichen pasture data with only continuous constraints. The data sets are taken into use with:[1]

[1] If you do not have the library, you must first install the required library from a repository using command `install.packages("vegan")`, or do the same from the menu entry.

```
R> library(vegan)
R> data(varespec)
R> 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:

```
R> str(varechem)
R> 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

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

It is 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:

```
R> 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:

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

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

```
R> varechem[2,]
R> 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:

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

3 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`:

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

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

For a full numerical survey, you can use `summary`. The `summary` lists scores for every species and every site – and for sites two different kind of scores. The output is long, but if you only want to get an idea of the style of the results, you can delimit the output to the `head` (or `tail`) of each type of scores:

```
R> head(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 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.

```
R> spenvcor(mm)
```

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

```
R> plot(mm)
```

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

```
R> 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:

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

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

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

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

4 Formula Interface

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

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

This indeed is the recommended method of defining the model, because it allows full control of the model and some further analyses are possible only when the model was defined through a formula.

With formula interface you can also use factor constraints:

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

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

4.1 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:

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

Then we can use `step` function to select the "best" model. The `step` function uses Akaike's Information 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 ("`-`" before their name) should be significant, and all excluded variables ("`+`" before their name) should be insignificant.[2]

```
R> 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 (if at all). 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:

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

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

The constrained ordination methods really do not have AIC, although it is calculated in `step`. Function `ordistep` uses significance test based on permutation instead of the non-existing AIC (see the next chapter).[3] The results of random permutations really are random, and therefore the model choice can have a random component. The usage is almost identical to the AIC based `step`:

```
R> m <- ordistep(m0, scope = formula(m1))
R> m$anova
```

One problem with model building is that constraining variables are not independent, but they are correlated. Any correlated variable can be explained with other variables. Such variables are redundant ("expendable") when they are with other variables, but they may be the best variables alone and prevent

[2] In Windows you should set off "buffered output" from the menu so that you can see the results as soon as they are calculated; with buffered output the all results are printed only after the whole analysis ends.

[3] `ordistep` is available from **vegan** version 1.17-0. If you have older **vegan** you can skip the following command (or install new **vegan**).

other variables from entering 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 your taste) are regarded as highly multicollinear (dependent on others). The VIF of a variable will depend on the set it sits with:

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

4.2 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:

```
R> m0 <- cca(dune ~  1, dune.env)
R> m1 <- cca(dune ~ ., dune.env)
R> m <- step(m0, scope=formula(m1), test="p")
R> 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 keeps the dependent variable (left hand side) and all previously entered constraints (hence "`. ~ .`"), and adds `Management` to the previous model:

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

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:

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

If needed, you can manually continue model building.

5 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:

```
R> anova(m)
```

We actually used function `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. This kind of functions that are automatically adapted to the result type are called *method functions*. We have already seen many of them: for instance `plot` and `summary` work differently for different types of results.

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 ($P = 0.05$ 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:

```
R> 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"):

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

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

```
R> 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).

6 Conditioned or partial ordination

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

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

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

```
R> 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`:

```
R> 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.

6.1 Partition of Variation

The basic partial ordination gives decomposition of the inertia into conditional (partialled out) and (residual) constrained component. The decomposition is sequential: conditions are partialled out before analysing constraints. Reversing the order of terms changes the components. Often we want to have a neutral decomposition of variation into unique and shared components of several sources. For this we can use function `varpart` of **vegan** which can handle upto four sources of variation.

Function `varpart` partitions adjusted R^2. Unlike ordinary R^2 or variance, adjusted R^2 is unbiased and its expected value is $R^2 = 0$ for random data (more in lectures). However, it can be easily calculated only for RDA and db-RDA. In principle, CCA can also be used, but calculations are lengthy and complicated, and no software is implemented for them.

The example above can be analysed using command:

```
R> mod <- varpart(dune, ~ Management, ~ A1 + Moisture, data = dune.env)
R> mod
```

The first argument (`dune`) gives the dependent data, and the next two to four arguments give the sources of variation. These can be single variables, matrices or one-sided model formulae like above.

The output can be tricky to read: names `X1`, `X2` etc refer to the original sources. These are divided into unique and shared components. In our example, the source `X1` (`Management`) has two components: unique component `a` and a component `b` that is shared with the source `X2`. Source `X2` consists of its unique component `c` and the same shared component `b`. Consquently, `X1` = `a` + `b` and `X2` = `b` + `c`. The unique component is the variation that can be explained only by the source and not by other source, and it is shown with operator `|` in the output: `a` = `X1 | X2`. The lower case letters can be deciphered using:

```
R> showvarparts(2)
R> showvarparts(3)
```

and the decomposition can be displayed in the same diagrams with

```
R> plot(mod)
```

The decomposition can be done by hand:

```
R> ab <- rda(dune ~ Management, dune.env)
R> bc <- rda(dune ~ A1 + Moisture, dune.env)
R> abc <- rda(dune ~ Management + A1 + Moisture, dune.env)
R> a <- rda(dune ~ Management + Condition(A1 + Moisture), dune.env)
R> c <- rda(dune ~ A1 + Moisture + Condition(Management), dune.env)
```

However, the shared component `b` cannot be found with a direct model. All the models above define testable models, but component `c` is non-testable and can be only found indirectly as a difference of the models. The components of variation can be negative for various reasons. Simplest reasons is that they are negative because of random variation in adjusted R^2, but there can be other, more complicated reasons (like multivariate non-linear dependence between sources of variation).