Multivariate Analysis of Ecological Communities in R

Jari Oksanen

March 16, 2005

Abstract

This tutorial demonstrates the use of basic ordination methods in R package vegan. The tutorial assumes basic familiarity both with R and with ordination methods. Package vegan supports all basic ordination method, including non-metric multidimensional scaling. The constrained ordination methods include constrained analysis of proximities, redundancy analysis and constrained correspondence analysis. Package vegan also has support functions for environmental fitting and ordination graphics. In addition to ordination methods, vegan contains several methods for analysis species diversity, but these methods are not discussed in this tutorial.

Contents

1 Introduction 2

2 Ordination: basic method 3

2.1 Non-metric Multidimensional scaling .................. 3
2.2 Comparing ordinations: Procrustes rotation .......... 4
2.3 Eigenvector methods ..................................... 5
2.4 Detrended correspondence analysis ..................... 8

3 Environmental interpretation 9

3.1 Vector fitting ............................................. 9
3.2 Surface fitting ........................................... 10
3.3 Factors .................................................... 12
3.4 Species as factors ....................................... 13

4 Constrained ordination 14

4.1 Model specification ....................................... 15
4.2 Permutation tests ........................................ 18
4.3 Model building ............................................ 18
4.4 Linear combinations and weighted averages ........ 24
4.5 Biplot arrows ............................................. 25
4.6 Conditioned or partial models ......................... 26
1 Introduction

This tutorial demonstrates typical work flows in multivariate ordination analysis of biological communities. The tutorial first discusses basic unconstrained analysis and environmental interpretation of their results. Then it introduces constrained ordination using constrained correspondence analysis as an example: alternative methods such as constrained analysis of proximities and redundancy analysis can be used (almost) similarly. Finally the tutorial describes analysis of species–environment relations without ordination, and elements of modifying ordination graphics.

The examples in this tutorial are tested: This is a Sweave document. The original source file contains only text and R commands: their output and graphics are generated while running the source through Sweave. However, you may need a recent version of vegan. This document was generated using version 1.7-52, but any recent version will repeat (most of) the analyses.

The manual covers ordination methods in vegan. It does not discuss many other methods in vegan. For instance, there are several functions for analysis of biodiversity: diversity indices (diversity, renyi, fisher.alpha), extrapolated species richness (specpool, estimateR), species accumulation curves (specaccum), species abundance models (radfit, fisherfit, prestonfit) etc. Neither is vegan the only R package for ecological community ordination. Base R has standard statistical tools, labdsv complements vegan with some advanced methods and provides alternative versions of some methods, and ade4 provides an alternative implementation for the whole gamme of ordination methods. Clustering and classification also are beyond the scope of this tutorial.

The tutorial explains only the most important methods and shows typical work flows. I see ordination primarily as a graphical tool, and I do not show too much exact numerical results. Instead, there are small vignettes of plotting results in the margins close to the place where you see a plot command. I suggest that you repeat the analysis, try different alternatives and inspect the results more thoroughly at your leisure. The functions are explained only briefly, and it is very useful to check the corresponding help pages for more thoroughly explanation of the methods. The methods also are only briefly explained. It is best to consult a textbook on ordination methods, or my lectures, for firmer theoretical background.
2 Ordination: basic method

2.1 Non-metric Multidimensional scaling

Non-metric multidimensional can be performed using `isoMDS` function in the MASS package. By default, this function needs dissimilarities as input. There are several alternative dissimilarity measures. Function `vegdist` in vegan contains some alternatives which are found good in community ecology. The default is Bray–Curtis dissimilarity, nowadays often known as Steinhaus dissimilarity, or in Finland as Sørensen index. The basic steps are:

```r
> library(vegan)
> library(MASS)
> data(varespec)
> vare.dis <- vegdist(varespec)
> vare.mds0 <- isoMDS(vare.dis)
```

```r
Initial value 18.026495
iter 5 value 10.095483
final value 10.020469
converged
```

The default is to find two dimensions, and use metric scaling (`cmdscale`) as the starting solution. The solution is iterative, as can be seen from the tracing information (this can be suppressed setting `trace = F`).

The results of `isoMDS` is a list (items `points`, `stress`) for the configuration and the stress. Stress $S$ is a statistic of goodness of fit, and it is a function of ordination distances $d$ and non-linear monotone transformation of observed dissimilarities $\theta(d)$. Functions `scores` and `ordiplot` in vegan can be used to handle the results:

```r
> ordiplot(vare.mds0, type = "t")
```

Only site scores were shown, because dissimilarities did not have information about species.

The iterative search is very difficult in NMDS, because of the nonlinear relationship between ordination and original dissimilarities. The iteration easily gets trapped into local optima instead of the global optimum. Therefore it is recommended to use several random starts, many ending in different solutions, and select among similar solutions with smallest stresses. This may be tedious, but vegan has function `metaMDS` which does this, and many more things. The tracing output is long, and we suppress it with `trace = 0`, but normally we want to see that something happens, since the analysis can take a long time:

```r
> vare.mds <- metaMDS(varespec, trace = FALSE)
> vare.mds
```

Call:
```
metaMDS(comm = varespec, trace = FALSE)
```
Nonmetric Multidimensional Scaling using isoMDS (MASS package)

Data:  \texttt{wisconsin(sqrt(varespec))}

Distance: \texttt{bray}

Dimensions: 2

Stress: 18.26

Two convergent solutions found after 20 tries

\texttt{> plot(vare.mds, type = "t")}

We did not calculate dissimilarities in a separate step, but we gave the original data matrix as input. The result is more complicated than previously, and has quite a few components in addition to those in isoMDS results: points, species, dims, stress, data, distance, converged, tries, call. The function wraps recommended procedures into one command. So what happened here?

1. The range of data values was so large that the data were square root transformed, and then submitted to Wisconsin double standardization, or species divided by their maxima, and stands standardized to equal totals. These two standardizations often improve the quality of ordinations, but we forgot to think about them in the initial analysis.

2. Function computed Bray–Curtis dissimilarities.

3. Function run isoMDS with several random starts, and stopped either after a certain number of tries, or after finding two similar configurations with minimum stress. In any case, it returned the best solution.

4. Function rotated the solution so that the largest variance of site scores will be on the first axis.

5. Function scaled the solution so that one unit corresponds to halving of community similarity from replicate similarity.

6. Function added species scores as weighted averages of site scores, but expanded them so that species and site scores have equal variance.

The help page for \texttt{metaMDS} will give more details, and point to explanation of functions used in the function.

2.2 Comparing ordinations: Procrustes rotation

Two ordinations can be very similar, but this may be difficult to see, because axes have slightly different orientations and different signs. Actually, in NMDs the sign, orientation, scale and location of the axes are not defined, although \texttt{metaMDS} uses simple method to fix the last two components. The best way to compare ordinations is to use Procrustes rotation. Procrustes rotation uses uniform scaling (expansion or contraction) and
rotation to minimize the squared differences between two ordinations. Package vegan has function procrustes to perform Procrustes analysis.

How much did we gain with using metaMDS instead of default isoMDS?

\[
\text{Call:}
\text{procrustes}(X = \text{vare.mds}, Y = \text{vare.mds0})
\]

Procrustes sum of squares: 0.156

> plot(pro, kind = 2)

In this case the differences were fairly small, and mainly concerned two points. You can use identify function to identify those points in an interactive session, or you can ask a plot of residual differences only:

> plot(pro, kind = 2)

The descriptive statistic is “Procrustes sum of squares” or the sum of squared arrows in the Procrustes plot. Procrustes analysis is nonsymmetric, and the statistic would change with reversing the order of ordinations in the call. With argument symmetric = TRUE, both solutions are first scaled to unit variance, and a more scale-independent and symmetric statistic is found (often known as Procrustes \( m^2 \)).

### 2.3 Eigenvector methods

Non-metric multidimensional scaling was a hard task, because any kind of dissimilarity measure could be used and dissimilarities were nonlinearly mapped into ordination. If we accept only certain types of dissimilarities and make a linear mapping, the ordination becomes a simple task of rotation and projection. In that case we can use eigenvector methods. Principal components analysis (PCA) and correspondence analysis (CA) are the most important eigenvector methods in community ordination. In addition, principal coordinates analysis a.k.a. metric multidimensional scaling is used occasionally.

Principal components analysis can be run as:

> vare.pca <- rda(varespec)
> vare.pca
2.3 Eigenvector methods

Call:
\[ \text{rda}(X = \text{varespec}) \]

Inertia Rank
Total 1826
Unconstrained 1826 23
Inertia is variance

Eigenvalues for unconstrained axes:
\[
\begin{array}{cccccccc}
\text{PC1} & \text{PC2} & \text{PC3} & \text{PC4} & \text{PC5} & \text{PC6} & \text{PC7} & \text{PC8} \\
983.0 & 464.3 & 132.3 & 73.9 & 48.4 & 37.0 & 25.7 & 19.7 \\
\end{array}
\]
(Showed only 8 of all 23 unconstrained eigenvalues)

The output tells that the total inertia is 1826, and the inertia is variance. The sum of all 23 (rank) eigenvalues would be equal to the total inertia. In other words, the solution decomposes the total variance to linear components. We can easily see that the variance equals inertia:

\[
> \text{sum}(\text{apply(varespec, 2, var)})
\]

[1] 1826

Function \text{apply} applies function \text{var} or variance to dimension 2 or columns (species), and then \text{sum} takes the sum of these values. Inertia is the sum of all species variances. The eigenvalues sum up to total inertia. In other words, they each “explain” a certain proportion of total variance. The first axis “explains” 983/1826 = 53.8 % of total variance.

The species ordination looks somewhat unsatisfactory: only reindeer lichens (\textit{Cladina}) and \textit{Pleurozium schreberi} are visible, and all other species are crowded at the origin. This happens because inertia was variance, and only abundant species with high variance are worth explaining. Standardizing all species to unit variance, or using the correlation coefficients instead of covariances will give a more balanced ordination:

\[
> \text{vare.pca} <- \text{rda(varespec, scale = TRUE)}
> \text{vare.pca}
\]

Call:
\[ \text{rda}(X = \text{varespec}, \text{scale} = \text{TRUE}) \]

Inertia Rank
Total 44
Unconstrained 44 23
Inertia is correlations

Eigenvalues for unconstrained axes:
\[
\begin{array}{cccccccc}
\text{PC1} & \text{PC2} & \text{PC3} & \text{PC4} & \text{PC5} & \text{PC6} & \text{PC7} & \text{PC8} \\
8.90 & 4.76 & 4.26 & 3.73 & 2.96 & 2.88 & 2.73 & 2.18 \\
\end{array}
\]
(Showed only 8 of all 23 unconstrained eigenvalues)
> plot(vare.pca, scaling = 3)

Now inertia is correlation, and the correlation of a variable with itself is one. Thus the total inertia is equal to the number of variables (species). The rank or the total number of eigenvectors is the same as previously. The maximum possible rank is defined by the dimensions of the data: it is one less than smaller of number of species or number of sites:

> dim(varespec)

[1] 24 44

If there are species or sites similar to each other, rank will be reduced even from this.

The percentage explained by the first axis decreased from the previous pca. This is natural, since previously we needed to “explain” only the abundant species with high variances, but now we have to explain all species equally. We should not look blindly at percentages, but the result we get.

Correspondence analysis is very similar to pca:

> vare.ca <- cca(varespec)
> vare.ca

Call:
cca(X = varespec)

Inertia Rank
Total  2.08
Unconstrained  2.08  23
Inertia is mean squared contingency coefficient

Eigenvalues for unconstrained axes:
CA1 CA2 CA3 CA4 CA5 CA6 CA7 CA8
0.525 0.357 0.234 0.195 0.178 0.122 0.115 0.089
(Showed only 8 of all 23 unconstrained eigenvalues)

> plot(vare.ca)

Now the inertia is called mean squared contingency coefficient. Correspondence analysis is based on Chi-squared distance, and the inertia is the Chi-squared statistic of a data matrix standardized to unit total:

> chisq.test(varespec/sum(varespec))

Pearson’s Chi-squared test

data: varespec/sum(varespec)
X-squared = 2.083, df = 989, p-value = 1

You should not pay any attention to P-values which are certainly misleading, but notice that the reported X-squared is equal to the inertia above.
Correspondence analysis is a weighted averaging method. In the graph above species scores were weighted averages of site scores. With different scaling of results, we could display the site scores as weighted averages of species scores:

```r
> plot(vare.ca, scaling = 1)
```

We already saw an example of `scaling = 3` or symmetric scaling in PCA. The other two integers mean that either species are weighted averages of sites (2) or sites are weighted averages of species (1). When we take a weighted average, the range of averages shrinks from the original values. The shrinkage factor is equal to the eigenvalue of `CA`, which has a theoretical maximum of 1.

### 2.4 Detrended correspondence analysis

Correspondence analysis was a much better and more robust method for community ordination than principal components analysis. However, with long ecological gradients it suffered from some drawbacks or “faults” which were corrected in detrended correspondence analysis (DCA):

- Single long gradients appeared as curves or arcs in ordination: the solution was to detrend the later axes by making their means equal along segments of previous axes.
- Sites were packed more closely at gradient extremes than at the centre: the solution was to rescale the axes to equal variances of species scores.
- Rare species seemed to have an unduly high influence on the results: the solution was to downweight rare species.

All these three separate tricks are incorporated in function `decorana` which is a faithful port of Mark Hill’s original programme with the same name. The usage is simple:

```r
> vare.dca <- decorana(varespec)
> vare.dca
```

Call:
```
decorana(veg = varespec)
```

Detrended correspondence analysis with 26 segments. Rescaling of axes with 4 iterations.

```
DCA1  DCA2  DCA3  DCA4
Eigenvalues 0.524 0.325 0.2001 0.1918
Decorana values 0.525 0.157 0.0967 0.0608
Axis lengths 2.816 2.205 1.5465 1.6486
```

```r
> plot(vare.dca)
```
Function `decorana` finds only four axes. Eigenvalues are defined as shrinkage values in weighted averages, similarly as in `cca` above. The “Decorana values” are the numbers that the original programme returns as “eigenvalues” — I have no idea of their possible meaning, and they should not be used. Most often people comment on axis lengths, which sometimes are called “gradient lengths”. The etymology is obscure: these are not gradients, but ordination axes. It is often said that if the axis length is shorter than two units, the data are linear, and PCA should be used. This is only folklore and not based on research which shows that CA is at least as good as PCA with short gradients, and usually better.

The current data set is homogeneous, and the effects of DCA are not very large. In heterogeneous data with a clear arc effect the changes often are more dramatic. Rescaling may have larger influence than detrending in many cases.

The default analysis is without downweighting of rare species: see help pages for the needed arguments. Actually, `downweight` is an independent function that can be used with `cca` as well.

There is a school of thought that regards DCA as the method of choice in unconstrained ordination. However, it seems to be a fragile and vague back of tricks that is better avoided.

## 3 Environmental interpretation

It is often possible to “explain” ordination using ecological knowledge on studied sites, or knowledge on the ecological characteristics of species. Usually it is preferable to use external environmental variables to interpret the ordination. There are many ways of overlaying environmental information onto ordination diagrams. One of the simplest is to change the size of plotting characters according to an environmental variables (argument `cex` in `plot` functions). The `vegan` package has some useful functions for fitting environmental variables.

### 3.1 Vector fitting

The most commonly used method of interpretation is to fit environmental vectors onto ordination. The fitted vectors are arrows with the interpretation:

- The arrow points to the direction of most rapid change in the the environmental variable. Often this is called as the direction of the gradient.
- The length of the arrow is proportional to the correlation between ordination and environmental variable. Often this is called the strength of the gradient.

Fitting environmental vectors is easy. The example uses the previous NMDS result and environmental variables in the data set `varechem`:

```r
> data(varechem)
> ef <- envfit(vare.mds, varechem, permu = 1000)
> ef
```
### 3.2 Surface fitting

Vector fitting is popular, and it provides a compact way of simultaneously displaying large number of environmental variables. However, it implies a linear relationship between ordination and environment: direction and strength are all you need to know. This may not always be appropriate.

Function `ordisurf` fits surfaces of environmental variables in two dimensional ordinations. It uses generalized additive models in function `gam` of package `mgcv`. Function `gam` can use thinplate splines in two dimensions, and automatically selects the degree of smoothing by generalized cross-validation. If the response really is linear and vectors are appropriate, the fitted surface is a plane whose gradient is parallel to the arrow.
and the fitted contours are equally spaced parallel lines perpendicular to
the arrow.

In the following example I introduce two new R features:

- Function \texttt{envfit} can be called with formula interface. In formula,
  the left-hand side gives the ordination result, then there is a special
  character tilde (\texttt{~}) followed by the names of fitted variables. In
  addition, we must define the name of the \texttt{data} containing the fitted
  variables.

- The variables in data frames are not visible to R session unless the
data frame is \texttt{attach}ed to the session. We may not want to make all
variables visible to the session, because there may be synonymous
names, and we may use wrong variables with the same name in
some analyses. We can use function \texttt{with} which makes the given
data frame visible only to the following command.

Now we are ready for the example. We make vector fitting for selected
variables and then add surfaces to these variables and show all in the
same plot.

```r
> ef <- envfit(vare.mds ~ Al + Ca, varechem)
> plot(vare.mds)
> plot(ef)
> tmp <- with(varechem, ordisurf(vare.mds, Al, add = TRUE))
```

Loading required package: mgcv
This is mgcv 1.1-8
Loading required package: akima

```r
> with(varechem, ordisurf(vare.mds, Ca, add = TRUE,
+   col = "green4")
```

Family: gaussian
Link function: identity

Formula:
y \sim \mathbf{s}(x_1, x_2, k = \text{knots})

Estimated degrees of freedom:
4.287 total = 5.287

GCV score: 41337

Function \texttt{ordisurf} returns the result of fitted \texttt{gam}. If we save that
result, like we did in the first fit with \texttt{Al}, we can use it for all further
analyses with \texttt{gam}, like statistical testing and prediction of new values.
For instance, \texttt{fitted(ef)} will give the actual fitted values for sites. We
also have an alternative three-dimensional plotting function in the \texttt{mgcv}
package:

```r
> vis.gam(tmp)
```
3.3 Factors

Class centroids are a natural choice for fitting factor variables. $R^2$ can be used as a goodness-of-fit statistic similarly as with vectors. The significance can be tested with permutations just like in vector fitting. Variables can be defined as factors in R, and they will be treated accordingly without any special tricks.

As an example, we shall inspect dune meadow data which has several class variables.

```r
> data(dune)
> data(dune.env)
> dune.ca <- cca(dune)
> ef <- envfit(dune.ca, dune.env, permutations = 1000)
> ef

***VECTORS

    CA1  CA2 r2  Pr(>r)
A1  0.9982 0.0606 0.31 0.042 *
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
P values based on 1000 permutations.

***FACTORS:

Centroids:

Moisture1 -0.75 -0.14
Moisture2 -0.47 -0.22
Moisture4  0.18 -0.73
Moisture5  1.11  0.57
ManagementBF -0.73 -0.14
ManagementHF -0.39 -0.30
ManagementNM  0.65  1.44
ManagementSF  0.34 -0.68
UseHayfield  0.29  0.65
UseHaypastu -0.07 -0.56
UsePasture   0.52  0.05
Manure0     0.65  1.44
Manure1     -0.46 -0.17
Manure2     -0.59 -0.36
Manure3     0.52 -0.32
Manure4     -0.21 -0.88

Goodness of fit:

    r2  Pr(>r)
Moisture  0.41 0.010 **
Management 0.44 0.003 **
Use       0.18 0.080 .
Manure    0.46 0.007 **
```
3 ENVIRONMENTAL INTERPRETATION

---

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
P values based on 1000 permutations.

> plot(dune.ca, display = "sites")
> plot(ef)

The names of factor centroids are formed by combining the name of
the factor and the name of the level. Now the Dims show the centroids
for the level, and the $R^2$ values are for the whole factor, just like
the significance test. The plot looks congested. We shall discuss controlling
graphics later in this document, but obviously not all factors are necessary
in interpretation.

Package vegan has several functions for graphical display of factors.
Function ordihull draws an enclosing convex hull for the items in a
class, ordispider combines items to their (weighted) class centroid, and
ordiellipse draws ellipses for class standard deviations, standard errors
or confidence areas. The example displays all these for Management type
in the previous ordination:

> plot(dune.ca, display = "sites", type = "p")
> with(dune.env, ordiellipse(dune.ca, Management, kind = "se",
+ conf = 0.95))
> with(dune.env, ordispider(dune.ca, Management, col = "blue"))
> with(dune.env, ordihull(dune.ca, Management, col = "blue",
+ lty = 2))

Correspondence analysis is a weighted ordination method, and vegan
functions envfit and ordisurf will do weighted fitting, unless the user
specifies equal weights.

3.4 Species as factors

Species are normally displayed as points in ordination space. This may be
appropriate if the location of the species optimum is a sufficient parameter
to describe species response. It is not a sufficient parameter if response
widths vary or there are interactions between axes.

We can use species as a factor using logical conditions such as Cal.vul
> 0: this is TRUE for sites where species is present, and FALSE for sites
where the species is absent. Then we can display this factor using the
graphical tools for factors environmental variables. Naturally, we must
make the species names visible to the R by attaching the data frame or
using with like in the example below. The following example displays the
occurrence of Calluna vulgaris in ca of lichen pastures with a convex hull.
For this we must change the species occurrences into a factor, and then
specify we want to have convex hull only for the case when the species is
present (show.groups = TRUE). The following plot adjusts the point size
(cex) to the abundance of Calluna:

> plot(vare.ca, type = "p", display = "sites")
> with(varespec, points(vare.ca, dis = "sites", cex = sqrt(Cal.vul),
+ pch = 16, col = "blue"))
> with(varespec, ordihull(vare.ca, Cal.vul > 0, show = TRUE))


For species centroids, we should give the abundances as weights to that the results are consistent with weighted averages:

```r
> with(varespec, ordispider(vare.ca, Cal.vul > 0, show = TRUE, + w = Cal.vul, col = "blue"))
> with(varespec, ordiellipse(vare.ca, Cal.vul > 0, + show = TRUE, w = Cal.vul))
```

We can also use `ordisurf` to fit response surfaces for species:

```r
> tmp <- with(varespec, ordisurf(vare.ca, Cal.vul, + add = TRUE, family = quasipoisson))
```

It also is possible to fit vectors to species with `envfit`, but it is difficult to imagine a situation where this makes sense.

## 4 Constrained ordination

In unconstrained ordination we first find the major compositional variation, and then relate this variation to observed environmental variation. In constrained ordination we do not want to display all or even most of the compositional variation, but only the variation that can be explained by the used environmental variables, or constraints. Constrained ordination is often known as “canonical” ordination, but this name is misleading: there is nothing particularly canonical in these methods (see your favorite Dictionary for the term). The name was taken into use, because there is one special statistical method, canonical correlations, but these indeed are canonical: they are correlations between two matrices regarded to be symmetrically dependent on each other. The constrained ordination is non-symmetric: we have “independent” variables or constraints and we have “dependent” variables or the community. Constrained ordination rather is related to multivariate linear models.

The `vegan` has three constrained ordination methods which all are constrained versions of basic ordination methods:

- Constrained analysis of proximities (CAP) in function `capscale` is related to metric multidimensional scaling (`cmdscale`). It can handle any dissimilarity measures, and performs a linear mapping.

- Redundancy analysis (RDA) in function `rda` is related to principal components analysis. It is based on Euclidean distances and performs linear mapping.

- Constrained correspondence analysis (CCA) in function `cca` is related correspondence analysis. It is based on Chi-squared distances and performs weighted linear mapping.

We have already used functions `rda` and `cca` for unconstrained ordination: they will perform the basic unconstrained method as a special case if constraints are not used.

All these three vegan functions are very similar. The following examples mainly use `cca`, but other methods can be used similarly. Actually,
the results are similarly structured, and they inherit properties from each other. For historical reasons, \texttt{cca} is the basic method, and \texttt{rda} inherits properties from it. Function \texttt{capscale} inherits directly from \texttt{rda}, and through this from \texttt{cca}. Many functions, such as \texttt{plot}, are common to all these methods, and there are specific functions only if the method deviates from its ancestor. In \texttt{vegan} version 1.7-52 the following class functions are defined for these methods:

- \texttt{cca}: alias, anova, coef, deviance, extractAIC, fitted, goodness, plot, points, predict, print, residuals, scores, summary, text, weights
- \texttt{rda}: coef, deviance, fitted, goodness, predict, summary, weights
- \texttt{capscale}: none.

Many of these methods are internal functions that users rarely need. These methods have a general hierarchy: if there is no method available for the function, the method of the previous level will be used. When there is no specific method for \texttt{capscale}, it will use \texttt{rda} methods whenever possible, and otherwise default to \texttt{cca} methods. Because \texttt{print} and \texttt{plot} exist only for \texttt{cca}, all functions will have a consistent look to the user.

### 4.1 Model specification

The recommended way of defining a constrained model is to use model formula. Formula has a special character \texttt{∼}, and on its left-hand side you give the name of the community data, and right-hand gives the equation for constraints. In addition, you should give the name of the data set where to find the constraints. This fits a \texttt{CCA} for \texttt{varespec} constrained by soil Al, K and P:

```r
> vare.cca <- cca(varespec ~ Al + P + K, varechem)
> vare.cca
Call:
cca(formula = varespec ~ Al + P + K, data = varechem)

Inertia Rank
Total 2.083
Constrained 0.644 3
Unconstrained 1.439 20

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1 CCA2 CCA3
0.362 0.170 0.113

Eigenvalues for unconstrained axes:
CA1 CA2 CA3 CA4 CA5 CA6 CA7 CA8
0.3500 0.2201 0.1851 0.1551 0.1351 0.1003 0.0773 0.0537
(Showed only 8 of all 20 unconstrained eigenvalues)
```
4.1 Model specification

The output is similar as before in unconstrained ordination. However, now the total inertia is decomposed into constrained and unconstrained components. There were three constraints, so the rank of constrained component is three. The rank of unconstrained component is now 20, when it used to be 23 in previous analysis. The rank is the same as the number of axes: you have 3 constrained axes and 20 unconstrained axes. In some cases, the ranks may be lower than the number of constraints: some of the used constraints are dependent on each other, and they are aliased in the analysis, and an informative message is printed with the result.

It is very common to calculate the proportion of constrained inertia from the total inertia. However, total inertia does not have a clear meaning in CCA, and the meaning of this proportion is just as obscure. In RDA this would be the proportion of variance (or correlation). This may have a clearer meaning, but even in this case most of the total inertia may be random noise. It may be better to concentrate on results instead of these proportions.

Basic plotting works just like earlier:

\[
> \text{plot(vare.cca)}
\]

Now the ordination diagram also has arrows for constraints. These have similar interpretation as fitted vectors: the arrows point to the direction of the gradient, and the length indicates the strength of the variable in this dimensionality of solution. The vectors will be of unit length in full rank solution, but they are projected to the plane used in the plot. There is also a primitive 3D plotting function (which needs user interaction for final graphs) that shows all arrows in full length:

\[
> \text{ordiplot3d(vare.cca, type = "h")}
\]

Loading required package: scatterplot3d

The formula interface works with factor variables as well:

\[
> \text{dune.cca <- cca(dune ~ Management, dune.env)}
> \text{plot(dune.cca)}
> \text{dune.cca}
\]

Call:
cca(formula = dune ~ Management, data = dune.env)

Inertia Rank
Total 2.115
Constrained 0.604 3
Unconstrained 1.511 16
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1 CCA2 CCA3
0.319 0.182 0.103
Factor variable Management had four levels BF, HF, NM, SF. Internally R expressed these four levels as three contrasts (sometimes called “dummy variables”). The applied contrasts look like this:

\[
\begin{array}{ccc}
\text{ManagementHF} & \text{ManagementNM} & \text{ManagementSF} \\
0 & 0 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{array}
\]

We do not need but three variables to express four levels: if there is number one in a column, the observation belongs to that level, and if there is a whole line of zeros, the observation must belong to the omitted level, or the first. The basic plot function displays class centroids instead of vectors for factors.

In addition to these ordinary factors, R also knows ordered factors. Variable Moisture in dune.env is defined as an ordered four-level factor. In this case the contrasts look different:

\[
\begin{array}{ccc}
\text{Moisture.L} & \text{Moisture.Q} & \text{Moisture.C} \\
1 & -0.6708 & 0.5 & -0.2236 \\
2 & -0.2236 & -0.5 & 0.6708 \\
4 & 0.2236 & -0.5 & -0.6708 \\
5 & 0.6708 & 0.5 & 0.2236 \\
\end{array}
\]

Now R uses polynomial contrasts: the linear term L is equal to treating Moisture as a continuous variable, and the quadratic Q and cubic C terms show the nonlinear features. There were four distinct levels, and the number of contrasts is one less, just like in ordinary contrasts. The ordination configuration, eigenvalues or rank are just the same if the factor is defined as unordered or ordered, but the presentation of the factor in results may change:

```r
> vare.cca <- cca(dune ~ Moisture, dune.env)
> plot(vare.cca)
```

Now plot shows both the centroids of factor levels, and the contrasts. If we could change the ordered factor to a continuous vector, only the linear effect arrow would be important. If the response to the variable is nonlinear, the quadratic (and cubic) arrows would be long as well.

I have explained only the simplest usage of the formula interface. The formula is very powerful in model specification: you can transform your contrasts within the formula, you can define interactions, you can use polynomial contrasts etc. However, models with interactions or polynomials may be difficult to interpret.
4.2 Permutation tests

The significance of all contrasts together can be assessed using permutation tests: the contrasts are permuted randomly and the model is refitted. When constrained inertia in permutations is nearly always lower than observed constrained inertia, we say that constraints are significant.

The easiest way of running permutation tests is to use the mock anova function in vegan:

```r
> anova(vare.cca)
```

Permutation test for cca under reduced model

```r
Model: cca(formula = dune ~ Moisture, data = dune.env)

Df Chisq F N.Perm Pr(>F)
Model 3 0.63 2.25 200 <0.005 ***
Residual 16 1.49

---

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```r
> tmp <- anova(vare.cca, step = 2, perm = 2)
```

The Model refers to the constrained component, and Residual to the unconstrained component of the ordination, Chisq is the corresponding inertia, and Df the corresponding rank. The test statistic $F$, or more correctly “pseudo-$F$” is defined as their ratio. You should not pay any attention to its numeric values or to the numbers of degrees of freedom, since this “pseudo-$F$” has nothing to do with the real $F$, and the only way to assess its “significance” is permutation. In simple models like the one studied here we could directly use inertia in testing, but the “pseudo-$F$” is needed in more complicated model including “partially” terms.

The number of permutations was not specified in the mock anova function. The function tries to be lazy: it continues permutations only as long as it is uncertain whether the final $P$-value will be below or above the critical value (usually $P = 0.05$). If the observed inertia is never reached in permutations, the function may stop after 200 permutations, and if it is very often exceeded, it may stop after 100 permutations. When we are close to the critical level, the permutations may continue to thousands. In this way the calculations are fast when this is possible, but they are continued longer in uncertain cases. If you want to have a fixed number of iterations, you must specify that in anova.cca call or directly use the underlying function permutest.cca

The anova function only tests all constraints simultaneously. We will discuss inspecting single constraints in conditional (partial) models.

4.3 Model building

It is very popular to perform constrained ordination using all available constraints simultaneously. Increasing the number of constraints actually means relaxing constraints: the ordination becomes more similar to the unconstrained one. When the rank of unconstrained component reduces
towards zero, the are absolutely no constraints. However, the relaxation of constraints often happens much earlier in first ordination axes. If we do not have strict constraints, it may be better to use unconstrained ordination with vector fitting (or surface fitting) than unconstrained ordination. This also allows detection of compositional variation for which we have not observed environmental variables. In constrained ordination it is best to reduce the number of constraints to just a few, say three to five.

I do not want to encourage using all possible environmental variables as constraints together. However, there still is a shortcut for that purpose in formula interface:

\[
\text{mod1} \leftarrow \text{cca(} \text{varespec} \sim ., \text{varechem)}
\]

\[
\text{Call:}
\text{cca(formula = varespec} \sim \text{N + P + K + Ca + Mg + S + Al + Fe +}
\text{Mn + Zn + Mo + Baresoil + Humdepth + pH, data = varechem)}
\]

<table>
<thead>
<tr>
<th>Inertia Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:

<table>
<thead>
<tr>
<th>CCA1</th>
<th>CCA2</th>
<th>CCA3</th>
<th>CCA4</th>
<th>CCA5</th>
<th>CCA6</th>
<th>CCA7</th>
<th>CCA8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.43887</td>
<td>0.29178</td>
<td>0.16285</td>
<td>0.14213</td>
<td>0.11795</td>
<td>0.08903</td>
<td>0.07029</td>
<td>0.05836</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CCA9</th>
<th>CCA10</th>
<th>CCA11</th>
<th>CCA12</th>
<th>CCA13</th>
<th>CCA14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03114</td>
<td>0.01329</td>
<td>0.00836</td>
<td>0.00654</td>
<td>0.00616</td>
<td>0.00473</td>
</tr>
</tbody>
</table>

Eigenvalues for unconstrained axes:

<table>
<thead>
<tr>
<th>CA1</th>
<th>CA2</th>
<th>CA3</th>
<th>CA4</th>
<th>CA5</th>
<th>CA6</th>
<th>CA7</th>
<th>CA8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.19776</td>
<td>0.14193</td>
<td>0.10117</td>
<td>0.07079</td>
<td>0.05330</td>
<td>0.03330</td>
<td>0.01887</td>
<td>0.01510</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CA9</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00949</td>
<td></td>
</tr>
</tbody>
</table>

This result probably is very similar to unconstrained ordination:

\[
> \text{plot(procrustes(cca(varespec), mod1))}
\]

For heuristic purposes we should reduce the number of constraints so that we get some kind of idea of important environmental variables. In principle, constrained ordination should be used with designed \textit{a priori} constraints only. All kind of automatic tools of model selection are dangerous. There may be several alternative models which are nearly equally good. Small changes in data can cause large changes in selected models. There may be no route to the best model with the adapted strategy. The model building has a history: one different step in the beginning may lead into wildly different final models.

After all these warnings, I show how \textit{vegan} can be used to automatically select constraints into model using standard \textit{R} function \texttt{step}. The
step uses Akaike's information criterion (AIC) as the selection criterion. AIC is a penalized goodness-of-fit measure: the goodness-of-fit is basically derived from the unconstrained inertia penalized by the rank of the constrained ordination. In principle AIC is based on log-Likelihood that constrained ordination do not have. However, a deviance function changes the unconstrained inertia to Chi-squared in cca or sum of squares in rda and capscale. Then this deviance is treated like sum of squares in Gaussian models. If we have only continuous (or 1 d.f.) terms, this is the same as selecting variables by their contributions to constrained eigenvalues (inertia). With factors the situation is more tricky, because the factors must be penalized by the degrees of freedom, and there is no way of knowing how high should the penalty be. The step function may still be useful in helping to gain insight into the data, but it should not be trusted blindly (or at all), but it should be regarded as an aid in proper model building. For further details, see help page of deviance.cca.

After this longish introduction the example: using step is much simpler than explaining how it works. We need to give the model we start with, and the scope of possible models inspected. For this we need another formula trick: formula with only 1 as the constraint defines an unconstrained model. We must define it like this so that we can add new terms to the initially unconstrained model. The scope must be given as a list of formulae, but we can extract those formulae from fitted models using function formula. The following example begins with an unconstrained model mod0 and steps towards the previously fitted maximum model mod1:

```r
> mod0 <- cca(varespec ~ 1, varechem)
> options(digits = 7)
> mod <- step(mod0, scope = list(lower = formula(mod0),
+     upper = formula(mod1)))
```

Start: AIC= 130.31
```
varespec ~ 1

Df  AIC
+ Al   1  128.61
+ Mn   1  128.95
+ Humdepth 1  129.24
+ Baresoil 1  129.77
+ Fe    1  129.79
+ P     1  130.03
+ Zn    1  130.30
<none> 130.31
+ Mg    1  130.35
+ K     1  130.37
+ Ca    1  130.43
+ pH    1  130.57
+ S     1  130.72
+ N     1  130.77
+ Mo    1  131.19
```
### Step: AIC= 128.61
varespec ~ Al

<table>
<thead>
<tr>
<th></th>
<th>Df</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ P</td>
<td>1</td>
<td>127.91</td>
</tr>
<tr>
<td>+ K</td>
<td>1</td>
<td>128.09</td>
</tr>
<tr>
<td>+ S</td>
<td>1</td>
<td>128.26</td>
</tr>
<tr>
<td>+ Zn</td>
<td>1</td>
<td>128.44</td>
</tr>
<tr>
<td>+ Mn</td>
<td>1</td>
<td>128.53</td>
</tr>
<tr>
<td>&lt;none&gt;</td>
<td>1</td>
<td>128.61</td>
</tr>
<tr>
<td>+ Mg</td>
<td>1</td>
<td>128.70</td>
</tr>
<tr>
<td>+ N</td>
<td>1</td>
<td>128.85</td>
</tr>
<tr>
<td>+ Baresoil</td>
<td>1</td>
<td>128.88</td>
</tr>
<tr>
<td>+ Ca</td>
<td>1</td>
<td>129.04</td>
</tr>
<tr>
<td>+ Humdepth</td>
<td>1</td>
<td>129.08</td>
</tr>
<tr>
<td>+ Mo</td>
<td>1</td>
<td>129.50</td>
</tr>
<tr>
<td>+ pH</td>
<td>1</td>
<td>129.63</td>
</tr>
<tr>
<td>+ Fe</td>
<td>1</td>
<td>130.02</td>
</tr>
<tr>
<td>- Al</td>
<td>1</td>
<td>130.31</td>
</tr>
</tbody>
</table>

### Step: AIC= 127.91
varespec ~ Al + P

<table>
<thead>
<tr>
<th></th>
<th>Df</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ K</td>
<td>1</td>
<td>127.44</td>
</tr>
<tr>
<td>&lt;none&gt;</td>
<td>1</td>
<td>127.91</td>
</tr>
<tr>
<td>+ Baresoil</td>
<td>1</td>
<td>127.99</td>
</tr>
<tr>
<td>+ N</td>
<td>1</td>
<td>128.11</td>
</tr>
<tr>
<td>+ S</td>
<td>1</td>
<td>128.36</td>
</tr>
<tr>
<td>+ Mn</td>
<td>1</td>
<td>128.44</td>
</tr>
<tr>
<td>+ Zn</td>
<td>1</td>
<td>128.51</td>
</tr>
<tr>
<td>+ Humdepth</td>
<td>1</td>
<td>128.56</td>
</tr>
<tr>
<td>- P</td>
<td>1</td>
<td>128.61</td>
</tr>
<tr>
<td>+ Mo</td>
<td>1</td>
<td>128.75</td>
</tr>
<tr>
<td>+ Mg</td>
<td>1</td>
<td>128.79</td>
</tr>
<tr>
<td>+ pH</td>
<td>1</td>
<td>128.82</td>
</tr>
<tr>
<td>+ Fe</td>
<td>1</td>
<td>129.28</td>
</tr>
<tr>
<td>+ Ca</td>
<td>1</td>
<td>129.36</td>
</tr>
<tr>
<td>- Al</td>
<td>1</td>
<td>130.03</td>
</tr>
</tbody>
</table>

### Step: AIC= 127.44
varespec ~ Al + P + K

<table>
<thead>
<tr>
<th></th>
<th>Df</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none&gt;</td>
<td>1</td>
<td>127.44</td>
</tr>
<tr>
<td>+ N</td>
<td>1</td>
<td>127.59</td>
</tr>
<tr>
<td>+ Baresoil</td>
<td>1</td>
<td>127.67</td>
</tr>
<tr>
<td>+ Zn</td>
<td>1</td>
<td>127.84</td>
</tr>
<tr>
<td>+ S</td>
<td>1</td>
<td>127.89</td>
</tr>
<tr>
<td>- K</td>
<td>1</td>
<td>127.91</td>
</tr>
</tbody>
</table>
4.3 Model building

+ Mo 1 127.92
- P 1 128.09
+ Mg 1 128.17
+ Mn 1 128.34
+ Humdepth 1 128.44
+ Fe 1 128.79
+ pH 1 128.81
+ Ca 1 128.89
- Al 1 130.14

> options(digits = 4)
> mod

Call:
cca(formula = varespec ~ Al + P + K, data = varechem)

Inertia Rank
Total 2.083
Constrained 0.644 3
Unconstrained 1.439 20

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1 CCA2 CCA3
0.362 0.170 0.113

Eigenvalues for unconstrained axes:
CA1 CA2 CA3 CA4 CA5 CA6 CA7 CA8
0.3500 0.2201 0.1851 0.1551 0.1351 0.1003 0.0773 0.0537
(Showed only 8 of all 20 unconstrained eigenvalues)

We ended up with the same familiar model we have been using all the time
(and now you know the reason why this model was used as an example).
The \text{AIC} was based on deviance, and penalty for each added parameter
was 2 per degree of freedom, and every step the \text{AIC} is evaluated for all
possible additions (+) and removals (−). The stepping stops when \text{{\it none}}
or the current model is at the top.

Model building with \text{step} is fragile: small changes in data can cause
large changes in model. The models have a history, and they are built
adding or removing only one parameter at time. There may be no route
to a good model in this way. The strategy of model building can change
the final model as well. If we start with the largest model (\text{mod1}), the
final model will be different:

> modb <- step(mod1, scope = list(lower = formula(mod0),
+ upper = formula(mod1)), trace = 0)
> modb

Call:
cca(formula = varespec ~ P + K + Mg + S + Mn + Mo + Baresoil +
         Humdepth, data = varechem)
Inertia Rank
Total 2.083
Constrained 1.117 8
Unconstrained 0.967 15
Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
CCA1 CCA2 CCA3 CCA4 CCA5 CCA6 CCA7 CCA8
0.4007 0.2488 0.1488 0.1266 0.0875 0.0661 0.0250 0.0130

Eigenvalues for unconstrained axes:
CA1 CA2 CA3 CA4 CA5 CA6 CA7 CA8
0.25821 0.18813 0.11927 0.10204 0.06085 0.04461 0.02782
CA9 CA10 CA11 CA12 CA13 CA14 CA15
0.02691 0.01646 0.01364 0.00823 0.00655 0.00365 0.00238

The AIC of this model is 127.89 which is higher than reached in forward selection (127.44). We suppressed tracing to save some pages of output, but step adds its history in the result:

> modb$anova

<table>
<thead>
<tr>
<th>Step</th>
<th>Df</th>
<th>Deviance</th>
<th>Resid. Df</th>
<th>Resid. Dev</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-15</td>
<td>1551</td>
<td>130.1</td>
</tr>
<tr>
<td>2</td>
<td>Fe</td>
<td>1</td>
<td>115.2</td>
<td>-14</td>
<td>1667 129.8</td>
</tr>
<tr>
<td>3</td>
<td>Al</td>
<td>1</td>
<td>106.0</td>
<td>-13</td>
<td>1773 129.3</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>1</td>
<td>117.5</td>
<td>-12</td>
<td>1890 128.8</td>
</tr>
<tr>
<td>5</td>
<td>pH</td>
<td>1</td>
<td>140.4</td>
<td>-11</td>
<td>2031 128.5</td>
</tr>
<tr>
<td>6</td>
<td>Ca</td>
<td>1</td>
<td>141.2</td>
<td>-10</td>
<td>2172 128.1</td>
</tr>
<tr>
<td>7</td>
<td>Zn</td>
<td>1</td>
<td>165.3</td>
<td>-9</td>
<td>2337 127.9</td>
</tr>
</tbody>
</table>

Variable Al was the first to selected into model in forward selection, but it was the second to removed in backward elimination. Variable Al is strongly correlated with many other explanatory variables. This is obvious when looking at the variance inflation factors (vif) in the full model mod1:

> vif.cca(mod1)

<table>
<thead>
<tr>
<th>N</th>
<th>P</th>
<th>K</th>
<th>Ca</th>
<th>Mg</th>
<th>S</th>
<th>Al</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fe</td>
<td>Mn</td>
<td>Zn</td>
<td>Mo</td>
<td>Ba</td>
<td>soil</td>
<td>Humdepth</td>
</tr>
</tbody>
</table>

A common rule of thumb is that vif > 10 indicates a variable that is strongly dependent on other variables and does not have independent information. On the other hand, it may not be that variable that should be removed, but alternatively some other variables may be removed. So the vifs were all modest in model found by forward selection:

> vif.cca(mod)

<table>
<thead>
<tr>
<th>Al</th>
<th>P</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.012</td>
<td>2.365</td>
<td>2.379</td>
</tr>
</tbody>
</table>
4.4 Linear combinations and weighted averages

There are two kinds of site scores in constrained ordinations:

1. Linear combination scores \( \text{lc} \) which are linear combinations of constraining variables.

2. Weighted averages scores \( \text{wa} \) which are weighted averages of species scores.

These two scores are as similar as possible, and their (weighted) correlation is called the species–environment correlation:

\[ > \text{spenvcor(mod)} \]

\[
\begin{array}{ccc}
\text{CCA1} & \text{CCA2} & \text{CCA3} \\
0.8555 & 0.8133 & 0.8793 \\
\end{array}
\]

These two scores are as similar as possible in constrained ordination. However, eigenvalue is a more appropriate measure of similarity between \( \text{lc} \) and \( \text{wa} \) scores. Correlation coefficient is very sensitive to single extreme values, like seems to happen in the example above where axis 3 has the “best” correlation simply because it has some extreme points.

The opinions are divided on using \( \text{lc} \) or \( \text{wa} \) as primary results in ordination graphics. The \texttt{vegan} package prefers \( \text{wa} \) scores, whereas the major commercial programme for \texttt{cca} prefers \( \text{lc} \) scores. The \texttt{vegan} package comes with a separate document ("vegan FAQ") which studies the issue in more detail, but I will briefly discuss the subject here also, and show how you can circumvent my decisions.

The practical reason to prefer \( \text{wa} \) scores is that they are more robust against random error in environmental variables. All ecological observations have random error, and therefore it is better to use scores that are resistant to this variation. Another point is that I see the \( \text{lc} \) as constraints: the scores are dependent only on environmental variables, and community composition does not influence them. The \( \text{wa} \) scores are based on community composition, but so that they are as similar to constraints as possible. This duality is particularly clear when using a single factor variable as constraint: the \( \text{lc} \) scores are constant within each level of the factor. The \( \text{wa} \) scores show how well we can predict the factor level from community composition.

The \texttt{vegan} package has a graphical function \texttt{ordispider} which (among other alternatives) will combine \( \text{wa} \) scores to the corresponding \( \text{lc} \) score.

With a single factor constraint:

\[ > \text{dune.cca} \leftarrow \text{cca(dune} \sim \text{Management, dune.env}) \]
\[ > \text{plot(dune.cca, display} = \text{c("lc", "wa"), type} = \text{"p"}) \]
\[ > \text{ordispider(dune.cca, col} = \text{"blue"}) \]

We can compare this result to discriminant analysis, so that \( \text{lc} \) scores give the predicted class centroids, and \( \text{wa} \) give the predicted values. For distinct classes, there is no overlap among groups. In general, the length of \texttt{ordispider} segments is a visual image of species–environment correlation.
4.5 Biplot arrows

Biplot arrows are an essential part of constrained ordination plots. The arrows are based on (weighted) correlation of LC scores and environmental variables. They are scaled to unit length in the constrained ordination of full rank. When these arrows are projected onto 2D ordination plot, they look shorter if they go off the plane.

In vegan the biplot arrows are always scaled similarly irrespective of scaling of sites or species. With default scaling = 2, the biplot arrows have optimal relation to sites, but with scaling = 1 they rather are related to species.

The standard interpretation of biplot arrows is that a site should be perpendicularly projected onto the arrow to get a prediction of the variable from the ordination. The arrow will start from the (weighted) mean of the environmental variable, and the values will increase towards the arrow head, and decrease from the (weighted) mean to the opposite direction. Then we still should figure out the unit of change.

All of this is doable, and equations are available in the literature. However, vegan provides function ordisurf which can be used to automate this task. Moreover, it can be used to check the linearity hypothesis of the projection method. Function ordisurf performs weighted fitting, and the model should be consistent with the one used in arrow fitting. Let us inspect the result of step function, where Aluminium was the most important of the three variables selected into the model. Now we should fit the model to the LC scores, just like the arrows:

```r
> plot(mod, display = c("bp", "wa", "lc"))
> ef <- with(varechem, ordisurf(mod, Al, display = "lc", + add = TRUE))
```

The results are not like we expected: we get curves instead of parallel lines perpendicular to the Al arrow. It seems that we cannot use linear projection in this case. Linear projection actually works, but only in the full constrained rank, or in three dimensions. When we project the multidimensional solution onto a plane, we get the distortion observed. Projections become unreliable as soon as we have more than two constrained axes — but sometimes they may work quite well. In this case, P would display a linear response surface, although it was less important than Al in model building.

We saved the results of surface fitting in ef, and we can use this to predict new values. fitted(ef) gives the actual fitted values for LC scores. These are close to real values. However, we are more interested in using the community data to predict the Aluminium concentration, and we must find predicted values for WA scores. Function ordisurf uses names x1 and x2 for axis dimensions, and we must make a data frame with same names as newdata for predict.gam:

```r
> wa <- scores(mod, display = "wa")
> pred <- predict(ef, newdata = data.frame(x1 = wa[, + 1], x2 = wa[, 2]))
```

25
4.6 Conditioned or partial models

The effect of some environmental variables can be removed from the ordination before constraining with other variables. The analysis is said to be conditioned on variables, or in other words, it is partial after removing variation caused by some variables. These conditioning variables typically are “random” or background variables, and their effect is removed from the analysis based on “fixed” or interesting variables.

In vegan, the formula for constrained ordination can contain a `Condition` which specifies the variable or variables whose effect is removed from the analysis before constraining with other variables. As an example, let us inspect what would be the effect of designed `Management` after removing the natural variation caused by `Moisture`:

```r
> dune.cca <- cca(dune ~ Management + Condition(Moisture),
   + dune.env)
> plot(dune.cca)
> dune.cca
```

```
Call: cca(formula = dune ~ Management + Condition(Moisture),
   data = dune.env)

Inertia Rank
Total 2.115
Conditional 0.628 3
Constrained 0.374 3
Unconstrained 1.113 13

Inertia is mean squared contingency coefficient

Eigenvalues for constrained axes:
   CCA1  CCA2  CCA3
  0.2278 0.0849 0.0614

Eigenvalues for unconstrained axes:
   CA1   CA2   CA3   CA4   CA5   CA6   CA7   CA8
 0.35040 0.15206 0.12508 0.10984 0.09221 0.07711 0.05944 0.04776
   CA9   CA10  CA11  CA12  CA13
 0.03696 0.02227 0.02070 0.01083 0.00825

Now the total inertia is decomposed into three components: inertia explained by conditions, inertia explained by constraints and the remaining unconstrained inertia. We previously fitted a model with `Management` as the only constraint, and in that case constrained inertia was clearly

The prediction does not look very good, despite good eigenvalues and “significant” results.
higher than now. It seems that different Management was practiced in different natural conditions, and the variation we previously attributed to Management may be due to natural variation.

We can perform permutation tests for Management in conditioned model, and Management alone:

```r
> anova(dune.cca, perm.max = 2000)
Permutation test for cca under reduced model
Model: cca(formula = dune ~ Management + Condition(Moisture), data = dune.env)
  Df Chisq F N.Perm Pr(>F)
Model  3 0.37 1.46 2000 0.049 *
Residual 13 1.11
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> anova(cca(dune ~ Management, dune.env))
Permutation test for cca under reduced model
Model: cca(formula = dune ~ Management, data = dune.env)
  Df Chisq F N.Perm Pr(>F)
Model  3 0.60 2.13  200 <0.005 ***
Residual 16 1.51
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Inspected alone, Management seemed to be very significant, but the situation is much less clear after removing the variation due to Moisture.

Conditioned or partial models are sometimes used for decomposition of inertia into various components attributed to different sets of environmental variables. In some cases this gives meaningful results, but the groups of environmental variables should be non-linearly independent for unbiased decomposition. If the groups of environmental variables have polynomial dependencies, some of the components of inertia may even become negative (that is impossible). That kind of higher-order dependencies are almost certain to appear with high number of variables and high number of groups.

The conditioned models can also be used to inspect the significance of adding new environmental variables to models. The following example demonstrate how to assess the significance of term K added to a model already containing terms Al and P:

```r
> vare.cca <- cca(varespec ~ Condition(Al + P) + K, + varechem)
> anova(vare.cca)
Permutation test for cca under reduced model
Model: cca(formula = varespec ~ Condition(Al + P) + K, data = varechem)
  Df Chisq F N.Perm Pr(>F)
```

27
5 Community and environment

We already discussed environmental interpretation of ordination and environmentally constrained ordination. These both reduce the variation into an ordination space, and mainly inspect only the first dimensions. Sometimes we may wish to analyse vegetation–environment relationships without ordination, or in full space. Typically these methods use the full dissimilarity matrix in analysis.

5.1 Analysis of similarities

Analysis of similarities (ANOSIM) inspects the effect of class variables (factors) to community. The analysis can be performed with vegan function anosim. The example studies again the impact of Management in dune vegetation:

```r
> dune.dis <- vegdist(dune)
> dune.ano <- with(dune.env, anosim(dune.dis, Management))
> plot(dune.ano)
> dune.ano
```

Call:
`anosim(dis = dune.dis, grouping = Management)`

ANOSIM statistic R: 0.258
Significance: 0.009

Based on 1000 permutations

\[
R = \frac{\bar{r}_B - \bar{r}_W}{N(N - 1)/4}
\]

ANOSIM is often represented as a non-parametric variant of analysis of variance. Basically, its statistic \(R\) is based on mean ranks of within group (\(\bar{r}_W\)) and between group (\(\bar{r}_B\)) dissimilarities, and scaled into range \(-1\ldots +1\), and \(R = 0\) indicating independence. The statistic uses only rank-order information, and is conceptually related to nmds. The significance is based on permutation tests.

It seems that ANOSIM is not very robust, but it is sensitive to different group sizes and within group heterogeneities. You may get a significant results with groups of same mean ranks but different within-group variances.

5.2 Mantel test

Mantel test compares two sets of dissimilarities. Basically, it is the correlation between dissimilarity entries. As there are \(N(N - 1)/2\) distinct
dissimilarities among only $N$ objects, normal significance tests are not applicable. Mantel developed asymptotic test statistics, but vegan function `mantel` uses permutation tests.

In this example we study how well the lichen pastures (`varespec`) correspond to the environment. We have already used vector fitting to the same solutions. However, the ordination and environment may be non-linearly related, and we try now with function `mantel`. We first perform a PCA of environmental variables, and then compute dissimilarities for first principal components. We use standard R function `prcomp`, but `princomp` or `rda` will work as well. Function `scores` in vegan will work similarly with all these methods. The following uses the same standardizations for community dissimilarities as previously used in `metaMDS`.

```r
> pc <- prcomp(varechem, scale = TRUE)
> pc <- scores(pc, display = "sites", choices = 1:4)
> edis <- vegdist(pc, method = "euclid")
> vare.dis <- vegdist(wisconsin(sqrt(varespec)))
> mantel(vare.dis, edis)
```

Mantel statistic based on Pearson's product-moment correlation

Call:
`mantel(xdis = vare.dis, ydis = edis)`

Mantel statistic $r$: 0.381
Significance: <0.001

Empirical upper confidence limits of $r$:
<table>
<thead>
<tr>
<th>90%</th>
<th>95%</th>
<th>97.5%</th>
<th>99%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.133</td>
<td>0.173</td>
<td>0.196</td>
<td>0.245</td>
</tr>
</tbody>
</table>

Based on 1000 permutations

We could use a selection of environmental variables in PCA, or we could use standardized environmental variables directly without PCA — tastes vary. Function `bioenv` gives an intriguing alternative for selecting optimal subsets for comparing ordination and environment. There also is a partial Mantel test where we can remove the influence of third set dissimilarities from the analysis, but its results often are difficult to interpret.

### 5.3 Protest: Procrustes test

Procrustes test or `protest` compares two ordinations using symmetric Procrustes analysis. It is an alternative to Mantel tests, but uses reduced space instead of complete dissimilarity matrices. We can repeat the previous analysis, but now with the solution of `metaMDS` and two first principal components of the environmental analysis:

```r
> pc <- scores(pc, choices = 1:2)
> pro <- protest(vare.mds, pc)
> plot(pro)
> pro
```
Call:
protest(X = vare.mds, Y = pc)

Correlation in a symmetric Procrustes rotation: 0.683
Significance: <0.001
Based on 1000 permutations.

The significance is assessed by permutation tests. The statistic is now a Procrustes correlation derived from the symmetric Procrustes residual \( m^2 \). The correlation is clearly higher than the Mantel correlation for the corresponding dissimilarities. In lower number of dimensions we remove noise from the data which may explain higher correlations (as well as different methods of calculating the correlation). However, both methods are about as significant. Significance often is not a significant concept: even small deviations from randomness may be highly significant in large data sets. Function protest provides graphical presentations (“Procrustes superimposition plot”) which may be more useful in evaluating the congruence between configurations.

PROTEST (as ordinary Procrustes analysis) is often used in assessing similarities between different community ordinations. This is known as analysis of congruence.

6 Classification

The vegan mainly is a package for ordination and diversity analysis, and there is only a scanty support to classification. There are several other R packages with more extensive classification functions. Among community ecological packages, labdsv by Dave Roberts is particularly strong in classification functions.

This chapter describes performing simple classification tasks in community ecology that are sufficient to many community ecologists. The chapter discusses hierarchic clustering only.

6.1 Cluster analysis

Hierarchic clustering can be performed using standard R function hclust. In addition, there are several other clustering packages, some of which may be compatible with hclust. Function hclust needs a dissimilarity matrix as input. Function vegdist in vegan is a drop-in replacement for dist function in base R and provides ecologically meaningful dissimilarity indices. Some other good indices are provided by dsvdis function in labdsv.

Function hclust provides several alternative clustering strategies. Among community ecologists, most popular are single linkage a.k.a. nearest neighbour, complete linkage a.k.a. furthest neighbour, and various brands of average linkage methods. These are best illustrated with examples:

```r
> dis <- vegdist(dune)
> clus <- hclust(dis, "single")
> plot(clus)
```
Some people prefer single linkage, because it is conceptually related to minimum spanning tree which nicely can be represented in ordinations, and it is able to find discontinuities in the data. However, single linkage is prone to chain data so that single-site clusters are joined to large clusters.

```r
> cluc <- hclust(dis, "complete")
> plot(cluc)
```

Some people prefer complete linkage because it makes compact clusters. However, this is in part an artefact of method: the clusters are not allowed to grow to include different sites, because the complete linkage criterion would be violated.

```r
> clua <- hclust(dis, "average")
> plot(clua)
```

Some people (I included) prefer average linkage clustering, because it seems to be a compromise between the previous two extremes, and more neutral in grouping. There are several alternative methods loosely connected to “average linkage” family. Ward’s method seems to be popular in publications. It approaches complete linkage in its attempt to minimize variances in agglomeration. The default "average" method is the one often known as UPGMA which is popular in old-time genetics.

All these clustering methods are agglomerative. They start with combining two most similar sites to each other. Then they proceed by combining points to points or to groups, or groups to groups. The fusion criteria vary. The left hand side of all graphs shows the scale of fusion. The scales vary among methods, but all are based on the same dissimilarities with range:

```r
> range(dis)
[1] 0.2273 1.0000
```

The first fusion is between the same two most similar sites in all examples, and at the same minimum dissimilarity. In complete linkage the last fusion combines the two most dissimilar sites, and it is at the maximum dissimilarity. In single linkage the fusion level always is at the smallest gap between groups, and the reported levels are much lower than with complete linkage. Average linkage makes fusions between group centre points, and its fusion levels are between the previous two trees. The estimated dissimilarity between two points is the level where they are fused in a tree. Function `cophenetic` finds this estimated dissimilarity from a tree for every pair of points – the name of the function reflects the history of clustering in numerical taxonomy. Cophenetic correlation measures the similarity between original dissimilarities and dissimilarities estimated from the tree. For our three example methods:

```r
> cor(dis, cophenetic(clus))
[1] 0.6602
> cor(dis, cophenetic(cluc))
```
6.2 Display and interpretation of classes

Cluster analysis performs a hierarchic clustering, and its results can be inspected at as many levels as there are points: the extremes are that every point is in its private cluster, or that all points belong to the same cluster. The user commonly wants to inspect clustering at a certain level, as non-hierarchic system of certain number of clusters. The flattening of the clustering happens by cutting the tree at some fusion level so that we get a desired number of clusters.

Base R provides function `rect.hclust` to visualize the cutting, and function `cutree` to make a classification vector with certain number of classes:

```r
> plot(cluc)
> rect.hclust(cluc, 3)
```

The classification vector can be used as any other factor variable. In community analysis, we commonly perform the clustering on sites using community composition. A natural way of inspecting the goodness of this community classification is to see how well it predicts external environmental variables that were not used in clustering. The only continuous variable in the Dune data is the thickness of the A1 horizon:

```r
> boxplot(A1 ~ grp, data = dune.env, notch = TRUE)
```

If we wish, we may use all normal statistical methods with factors, such as functions `lm` or `aov` for formal testing of “signficance” of clusters. Classification can be compared against external factor variables as well. However, `vegan` does not provide any tools for this. It may be best to see the `labdsv` package and its tutorial for this purpose.

The clustering results can be displayed in ordination diagrams. All usual `vegan` functions for factors can be used: `ordihull`, `ordispider`, and `ordiellipse`. We shall see only the first as an example:

```r
> ord <- cca(dune)

> plot(ord, display = "sites")
> ordihull(ord, grp, lty = 2, col = "red")
```
It is said sometimes that overlaying classification in ordination can be used as a cross-check: if the clusters look distinct in the ordination diagram, (both) analyses probably were adequate. However, the classes can overlap and the analyses can still be good. It may be that you need three or more axes to display the multivariate class structure. In addition, ordination and classification may use different criteria. In our example, CA uses weighted Chi-squared criteria, and the clustering uses Bray-Curtis dissimilarities which may be quite different.

The vegan package has function `ordicluster` to overlay hclust tree in an ordination:

```r
> plot(ord, display = "sites")
> ordihull(ord, grp, lty = 2, col = "red")
> ordicluster(ord, cluc, prune = 2, col = "blue")
```

The function combines points and cluster midpoints similarly as in the original cluster dendrogram.

Single linkage clustering is the method most often used in connection with ordination diagram. Single linkage clustering is special among the clustering algorithms, because it actually combines points to points: it is only the nearest point that is recognized and no information on its cluster membership is used. The dendrogram, however, hides this information: it only shows the joins to clusters, but does not show which were the actual points that were joined. The tree connecting the individual points is called a minimum spanning tree (MST). In graph theory, ‘tree’ is a connected graph with no loops, ‘spanning tree’ is tree that connects all points, and minimum spanning tree is the one where the total length of connecting segments is shortest. In principle, single linkage clustering finds this tree, but it is hidden in the analysis. Function `spantree` finds this tree, and `ordispantree` displays the tree in vegan:

```r
> mst <- spantree(dis, toolong = 1)
> plot(ord, display = "sites", type = "n")
> ordispantree(ord, mst, col = "red", lwd = 2)
> points(ord, display = "sites", pch = 21, col = "red", +
       bg = "yellow")
```

In our dissimilarity index, distance = 1 means that there is nothing in common with two sample plots. Function `spantree` regards these maximum dissimilarities as missing data, and does not use them in building the tree. If all points cannot be connected because of these missing values, the result will consist of disconnected spanning trees. In graph theory this kind of pattern is known as a ‘forest’. MST is used sometimes to cross-check ordination: if the tree is linear, the ordination might be good. A curve tree may indicate arc or horseshoe artefacts, and a messy tree a bad ordination, or a need of higher number of dimensions. However, the results often are difficult to interpret, and MST usually is fairly useless.

### 6.3 Classified community tables

The aim of classification often is to make a classified community table. For this purpose, both sites and species should be arranged so that the
table looks structured. The original clustering may not be ideally structured, because the ordering of sites is not strictly defined in the cluster dendrogram. You can take any branch and rotate it around its base, and the clustering is the same. The tree drawing algorithms use heuristic rules to make the tree look aesthetically pleasing, but this same ordering may not be the best one for a structured community table.

Base R has a general tree class called `dendrogram` which is intended as a common base for any tree-like presentations. This class has a function to reorder a tree according to some external variable. The `hclust` result can be changed into `dendrogram` with function `as.dendrogram`, and this can be reordered with function `reorder`. The only continuous variable in the Dune data is the thickness of A1 horizon, and this could be used to arrange the tree. However, for a nicely structured community table we use another trick: `ca` is an ordination method that structures table optimally into a diagonal structure, and we can use its first axis to reorder the tree:

```r
> wa <- scores(ord, display = "sites", choices = 1)
> den <- as.dendrogram(clua)
> oden <- reorder(den, wa, mean)
```

The results really change, and it may take some effort to see that these two trees really are identical, except for the order of leaves.

```r
> op <- par(mfrow = c(2, 1))
> plot(den)
> plot(oden)
> par(op)
```

Function `vegemite` in `vegan` produces compact vegetation tables. It can take an argument `use` that is used to arrange the sites (and species, if possible). This argument can be a vector used to arrange sites, or it can be an ordination result, or it can be `hclust` result or a `dendrogram` object.

```r
> vegemite(dune, use = oden, zero = "-"
```

```
 XX X XX XXXXX
111111X11111111111
79157602183498234506
Airpra 23-----------------
Empnig -2-----------------
Hyprad 25---------------
Antodo 44-4234------------
Tripra ---225------------
Achmil 2-122243------------
Plalan 2--5553-33----------
Rumace ---536------2-2-----
Brohor ---22-44--3------
Lolper --726665726524-----
```

1From vegan version 1.7-52.
The **dendrogram** had no information on species, but it uses weighted averages to arrange them similarly as sites. This may not be optimal for a clustering results, but if the clusters are reordered nicely, the results may be very satisfactory with a nicely structured community table.

The **vegemite** output is very compact (hence the name), and it uses only one column for sites. In this case this was automatic, since Dune meadow data uses class scales. Percent cover scale can be transformed to traditional class scales, such as Braun-Blanquet, Domin or Hult-Sernander-Du Rietz.