Vegan: an introduction to ordination

Jari Oksanen

processed with vegan 2.7-3 in R version 4.5.2 (2025-10-31) on March 3, 2026

Abstract

The document describes typical, simple work pathways of vegetation ordination. Unconstrained ordination uses as examples detrended correspondence analysis and non-metric multidimensional scaling, and shows how to interpret their results by fitting environmental vectors and factors or smooth environmental surfaces to the graph. The basic plotting command, and more advanced plotting commands for congested plots are also discussed, as well as adding items such as ellipses, convex hulls, and other items for classes. The constrained ordination uses constrained (canonical) correspondence analysis as an example. It is first shown how a model is defined, then the document discusses model building and significance tests of the whole analysis, single constraints and axes.

Contents

Vegan is a package for community ecologists. This documents explains how the commonly used ordination methods can be performed in **vegan**. The document only is a very basic introduction. The current document only describes a small part of all **vegan** functions. For most functions, the canonical references are the **vegan** help pages.

1 Ordination

The **vegan** package contains all common ordination methods: Principal component analysis (functions `pca` and `rda`, or `prcomp` in the base `R`), correspondence

analysis (`ca`, `cca`), detrended correspondence analysis (`decorana`), metric scaling, also known as principal coordinate analysis (`pco`, `wcmdscale`, or `cmdscale` in base R), non-metric multidimensional scaling (`monoMDS`) with wrapper for common pipeline of use (`metaMDS`). Functions `rda` and `cca` mainly are designed for constrained ordination, and will be discussed later. In this chapter I describe functions `decorana` and `metaMDS`.

1.1 Detrended correspondence analysis

Detrended correspondence analysis (DCA) is done like this:

```
> library(vegan)
> data(dune)
> ord <- decorana(dune)
```

This saves ordination results in `ord`:

```
> ord

Call:
decorana(veg = dune)

Detrended correspondence analysis with 26 segments.
Rescaling of axes with 4 iterations.
Total inertia (scaled Chi-square): 2.1153

                        DCA1   DCA2    DCA3    DCA4
Eigenvalues           0.5117 0.3036 0.12125 0.14267
Additive Eigenvalues  0.5117 0.2985 0.12242 0.12984
Decorana values       0.5360 0.2869 0.08136 0.04814
Axis lengths          3.7004 3.1166 1.30055 1.47888
```

The display of results is very brief: only eigenvalues and used options are listed. Actual ordination results are not shown, but you can extract them with command `scores(ord)`. The `plot` function also automatically knows how to access the scores.

1.2 Non-metric multidimensional scaling

Function `metaMDS` is a bit special case. The actual ordination is performed by **vegan** function `monoMDS`. Function `metaMDS` is a wrapper to perform non-metric multidimensional scaling (NMDS) like recommended in community ordination: it uses adequate dissimilarity measures (function `vegdist`), then it runs NMDS several times with random starting configurations, compares results (function `procrustes`), and stops after finding twice a similar minimum stress solution. Finally it scales and rotates the solution, and adds species scores to the configuration as weighted averages (function `wascores`):

```
> ord <- metaMDS(dune, trace = FALSE)
> ord

Call:
metaMDS(comm = dune, trace = FALSE)

global Multidimensional Scaling using monoMDS
```

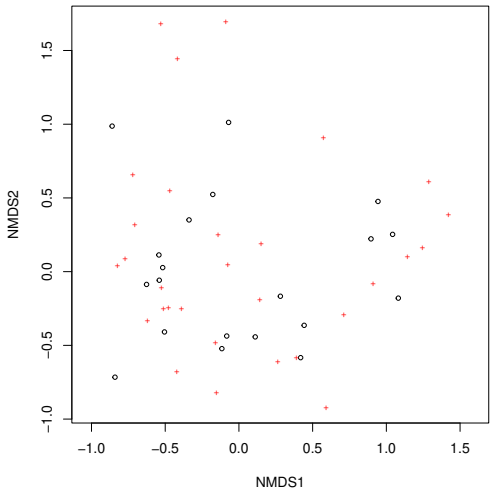

Figure 1: Default ordination plot.

```
Data:     dune
Distance: bray

Dimensions: 2
Stress:     0.1183186
Stress type 1, weak ties
Best solution was repeated 4 times in 20 tries
The best solution was from try 7 (random start)
Scaling: centring, PC rotation, halfchange scaling
Species: expanded scores based on 'dune'
```

2 Ordination graphics

Ordination is nothing but a way of drawing graphs, and it is best to inspect ordinations only graphically (which also implies that they should not be taken too seriously).

All ordination results of **vegan** can be displayed with a `plot` command (Fig. 1):

```
> plot(ord)
```

Default `plot` command uses either black circles for sites and red pluses for species, or black and red text for sites and species, resp. The choices depend on the number of items in the plot and ordination method. You can override the default choice by setting `type = "p"` for points, or `type = "t"` for text. For a better control of ordination graphics you can first draw an empty plot (`type = "n"`) and then add species and sites separately using `points` or `text` functions. In this way you can combine points and text, and you can select colours and character sizes freely (Fig. 2). The easiest way is to build the plot by layers using pipe (`|>`).

```
> plot(ord, type = "n") |>
      points("sites", cex = 0.8, pch=21, col="red", bg="yellow") |>
      text("species", cex=0.7, col="blue")
```

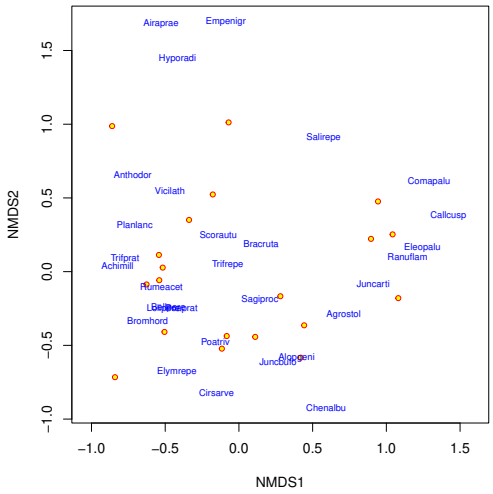

Figure 2: A more colourful ordination plot where sites are points and species are text.

All **vegan** ordination methods have a specific `plot` function. In addition, **vegan** has an alternative plotting function `ordiplot` that also knows many non-**vegan** ordination methods, such as `prcomp` and `cmdscale`. All **vegan** plot functions return invisibly an `ordiplot` object, so that you can use `ordiplot` support functions with the results (`points`, `text`, `identify`).

Alternative plotting methods are available in two packages:

- **vegan3d** on CRAN provides static 3D plots (`ordiplot3d`), dynamic 3D plots that can be spinned around or zoomed (`ordirgl`) and interactive 2D plots that can be edited moving point labels to better position (`orditkplot`).
- **ggvegan** on CRAN provides **ggplot2** graphics for most **vegan** objects. The **vegan** `scores` functions for ordination objects (and some others) recognize argument `tidy` which can be used to extract scores for **ggplot2** graphics. (There are some **lattice** graphics functions in **vegan**, but the plan is to phase out these in favour of **ggvegan** graphics.)

2.1 Cluttered plots

Ordination plots are often congested: there is a large number of sites and species, and it may be impossible to display all clearly. In particular, two or more species may have identical scores and are plotted over each other. Here some methods you can try:

- Use only `points` if you do not need to identify the item (and function `identify` can be used to add labels some of the points).
- Most **vegan** `plot` and `text` functions know arguments `optimize` and `bg`. With `optimize=TRUE` the exact scores are shown with labelled points, and the labels are positioned to minimize over-plotting. Argument `bg` gives the background colour of labels. With background you cannot see the text or points below the label, but you can read at least the uppermost text. These arguments can be used together. These two arguments often help with moderate cluttering.

- Zoom into graph setting axis limits `xlim` and `ylim`. You must typically set both, because **vegan** will maintain equal aspect ratio of axes.
- Use points and add label only to some points with `identify` command.
- Use `select` argument in ordination `text` and `points` functions to only show the specified items, possibly combined with argument `labels` for shorter names.
- Use automatic `orditorp` function that uses text only if this can be done without overwriting previous labels, but points in other cases.
- Use interactive `orditkplot` function in **vegan3d** that draws both points and labels for ordination scores, and allows you to drag labels to better positions. You can export the edited graph in several graphical formats, or return the edited positions to R for further processing.

2.2 Adding items to ordination plots

Vegan has a group of functions for adding information about classification or grouping of points onto ordination diagrams. Function `ordihull` adds convex hulls, `ordiellipse` adds ellipses enclosing all points in the group (ellipsoid hulls) or ellipses of standard deviation, standard error or confidence areas, `ordibar` draws a cross corresponding to the principal axes of ellipses, and `ordispider` combines items to their centroid (Fig. 3):

```
> data(dune.env)
> attach(dune.env)

> plot(ord, disp="sites", type="n")
> ordihull(ord, Management, col=1:4, lwd=3)
> ordiellipse(ord, Management, col=1:4, kind = "ehull", lwd=3)
> ordiellipse(ord, Management, col=1:4, draw="polygon")
> ordispider(ord, Management, col=1:4, label = TRUE)
> points(ord, disp="sites", pch=21, col="red", bg="yellow", cex=1.3)
```

In addition, you can overlay a cluster dendrogram from `hclust` using `ordicluster` or a minimum spanning tree from `spantree` with its `lines` function. Segmented arrows can be added with `ordiarrows`, lines with `ordisegments` and regular grids with `ordigrid`.

3 Fitting environmental variables

Vegan provides two functions for fitting environmental variables onto ordination:

- `envfit` fits vectors of continuous variables and centroids of levels of class variables (defined as `factor` in R). The arrow shows the direction of the (increasing) gradient, and the length of the arrow is proportional to the correlation between the variable and the ordination.
- `ordisurf` (which requires package **mgcv**) fits smooth surfaces for continuous variables onto ordination using thinplate splines with cross-validatory selection of smoothness.

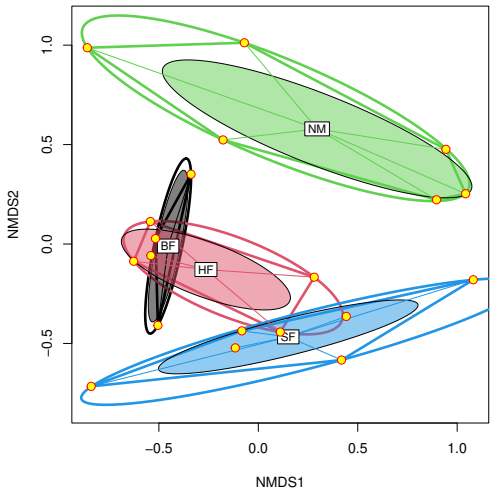

Figure 3: Convex hull, ellipsoid hull, standard error ellipse and a spider web diagram for Management levels in ordination.

Function `envfit` can be called with a `formula` interface, and it optionally can assess the "significance" of the variables using permutation tests:

```
> ord.fit <- envfit(ord ~ A1 + Management, data=dune.env, perm=999)
> ord.fit

***VECTORS

     NMDS1   NMDS2     r2 Pr(>r)
A1 0.96473 0.26323 0.3649  0.022 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Permutation: free
Number of permutations: 999

***FACTORS:

Centroids:
               NMDS1   NMDS2
ManagementBF -0.4534 -0.0102
ManagementHF -0.2636 -0.1282
ManagementNM  0.2957  0.5790
ManagementSF  0.1506 -0.4670

Goodness of fit:
               r2 Pr(>r)
Management 0.4134   0.01 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Permutation: free
Number of permutations: 999
```

The result can be drawn directly or added to an ordination diagram (Fig. 4):

```
> plot(ord, dis="site")
> plot(ord.fit, bg = "yellow")
```

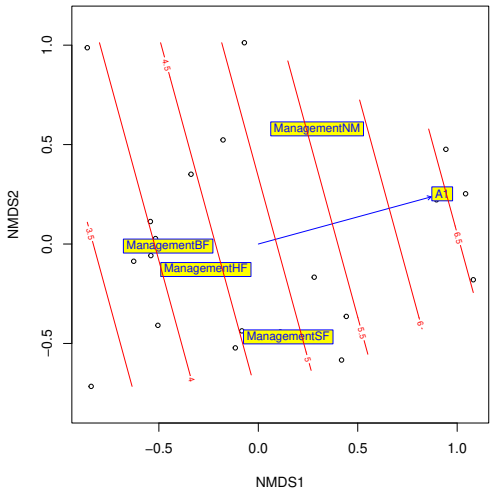

Figure 4: Fitted vector and smooth surface for the thickness of A1 horizon (`A1`, in cm), and centroids of Management levels.

Function `ordisurf` directly adds a fitted surface onto ordination, and it returns the result of the fitted `gam` (Fig. 4):

```
> ordisurf(ord, A1, add=TRUE)

Family: gaussian
Link function: identity

Formula:
y ~ s(x1, x2, k = 10, bs = "tp", fx = FALSE)

Estimated degrees of freedom:
1.59  total = 2.59

REML score: 41.58727
```

4 Constrained ordination

Vegan has three methods of constrained ordination: constrained or "canonical" correspondence analysis (function `cca`), redundancy analysis (function `rda`) and distance-based redundancy analysis (function `dbrda`). All these functions can have a conditioning term that is "partialled out". I only demonstrate `cca`, but all functions accept similar commands and can be used in the same way.

The preferred way is to use `formula` interface, where the left hand side gives the community data frame and the right hand side lists the constraining variables:

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

Call: cca(formula = dune ~ A1 + Management, data = dune.env)

              Inertia Proportion Rank
Total          2.1153     1.0000
Constrained    0.7798     0.3686    4
```

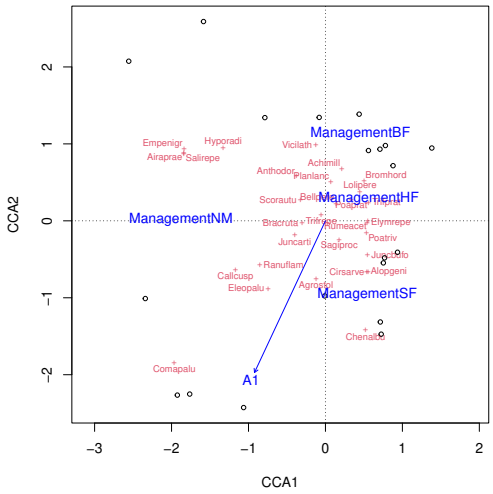

Figure 5: Plot of constrained correspondence analysis showing sites as points and with optimized location for species labels.

```
Unconstrained  1.3355     0.6314   15

Inertia is scaled Chi-square

Eigenvalues for constrained axes:
  CCA1   CCA2   CCA3   CCA4
0.3187 0.2372 0.1322 0.0917

Eigenvalues for unconstrained axes:
   CA1    CA2    CA3    CA4    CA5    CA6    CA7    CA8    CA9   CA10
0.3620 0.2029 0.1527 0.1345 0.1110 0.0800 0.0767 0.0553 0.0444 0.0415
  CA11   CA12   CA13   CA14   CA15
0.0317 0.0178 0.0116 0.0087 0.0047
```

The results can be plotted with (Fig. 5):

```
> plot(ord, spe.par = list(optimize = TRUE), sit.par = list(type="p"))
```

There are three groups of items: sites, species and centroids (and biplot arrows) of environmental variables. All these can be added individually to an empty plot with pipes, and all previously explained tricks of controlling graphics still apply. If only small changes are wanted to default settings, it is easier to change those parameters with a list of new argument values like in the example.

It is not recommended to perform constrained ordination with all available environmental variables: adding the number of constraints means slacker constraint, and you finally end up with solution similar to unconstrained ordination. In that case it is better to use unconstrained ordination with environmental fitting. However, if you really want to do so, it is possible with the following shortcut in `formula`:

```
> cca(dune ~ ., data=dune.env)

Call: cca(formula = dune ~ A1 + Moisture + Management + Use +
Manure, data = dune.env)

              Inertia Proportion Rank
Total          2.1153     1.0000
```

```
Constrained    1.5032     0.7106   12
Unconstrained  0.6121     0.2894    7

Inertia is scaled Chi-square

-- NOTE:
Some constraints or conditions were aliased because they were
redundant. This can happen if terms are constant or linearly
dependent (collinear): 'Manure^4'

Eigenvalues for constrained axes:
  CCA1   CCA2   CCA3   CCA4   CCA5   CCA6   CCA7   CCA8   CCA9  CCA10
0.4671 0.3410 0.1761 0.1532 0.0953 0.0703 0.0589 0.0499 0.0318 0.0260
 CCA11  CCA12
0.0228 0.0108

Eigenvalues for unconstrained axes:
    CA1     CA2     CA3     CA4     CA5     CA6     CA7
0.27237 0.10876 0.08975 0.06305 0.03489 0.02529 0.01798
```

The model gave a message that some constraints were aliased because they were redundant. This means that the variable did not have unique explanatory power, but it can be expressed with the help of other variables. Such redundant variables are not shown in ordination. In this case fourth degree polynomial of `Manure` (an ordered factor) was redundant ana aliased. There is one `Manure` level (0) which only occurs in `Management` level `NM` (natural management), and we know that `Manure` level once we know the `Management`.

4.1 Significance tests

vegan provides permutation tests for the significance of constraints. The test mimics standard analysis of variance function (`anova`), and the default test analyses all constraints simultaneously:

```
> anova(ord)

Permutation test for cca under reduced model
Permutation: free
Number of permutations: 999

Model: cca(formula = dune ~ A1 + Management, data = dune.env)
         Df ChiSquare      F Pr(>F)
Model     4   0.77978 2.1896  0.001 ***
Residual 15   1.33549
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

It is also possible to analyse terms separately:

```
> anova(ord, by="term")

Permutation test for cca under reduced model
Terms added sequentially (first to last)
```

```
Permutation: free
Number of permutations: 999

Model: cca(formula = dune ~ A1 + Management, data = dune.env)
           Df ChiSquare      F Pr(>F)
A1          1   0.22476 2.5245  0.015 *
Management  3   0.55502 2.0780  0.003 **
Residual   15   1.33549
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This test is sequential: the terms are analysed in the order they happen to be in the model. You can also analyse significances of marginal effects ("Type III effects"):

```
> anova(ord, by="margin")

Permutation test for cca under reduced model
Marginal effects of terms
Permutation: free
Number of permutations: 999

Model: cca(formula = dune ~ A1 + Management, data = dune.env)
           Df ChiSquare      F Pr(>F)
A1          1   0.17594 1.9761  0.035 *
Management  3   0.55502 2.0780  0.002 **
Residual   15   1.33549
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

4.2 Conditioned or partial ordination

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

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

Call: cca(formula = dune ~ A1 + Management + Condition(Moisture),
data = dune.env)

              Inertia Proportion Rank
Total          2.1153     1.0000
Conditional    0.6283     0.2970    3
Constrained    0.5109     0.2415    4
Unconstrained  0.9761     0.4615   12

Inertia is scaled Chi-square

Eigenvalues for constrained axes:
   CCA1    CCA2    CCA3    CCA4
0.24932 0.12090 0.08160 0.05904

Eigenvalues for unconstrained axes:
    CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8     CA9
0.30637 0.13191 0.11516 0.10947 0.07724 0.07575 0.04871 0.03758 0.03106
```

```
   CA10    CA11    CA12
0.02102 0.01254 0.00928
```

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

```
> anova(ord, by="term")
```

```
Permutation test for cca under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

Model: cca(formula = dune ~ A1 + Management + Condition(Moisture), data = dune.env)
           Df ChiSquare      F Pr(>F)
A1          1   0.11543 1.4190  0.106
Management  3   0.39543 1.6205  0.015 *
Residual   12   0.97610
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

If we had a designed experiment, we may wish to restrict the permutations so that the observations only are permuted within levels of `Moisture`. Restricted permutation is based on the powerful **permute** package. Function `how()` can be used to define permutation schemes. In the following, we set the levels with `plots` argument:

```
> how <- how(nperm=999, plots = Plots(strata=dune.env$Moisture))
> anova(ord, by="term", permutations = how)
```

```
Permutation test for cca under reduced model
Terms added sequentially (first to last)
Plots: dune.env$Moisture, plot permutation: none
Permutation: free
Number of permutations: 999

Model: cca(formula = dune ~ A1 + Management + Condition(Moisture), data = dune.env)
           Df ChiSquare      F Pr(>F)
A1          1   0.11543 1.4190  0.266
Management  3   0.39543 1.6205  0.004 **
Residual   12   0.97610
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```