Unconstrained ordination

Gavin Simpson

November 22, 2022

Today's topics

Today's topics

- Ordination
  - Principal Components Analysis (PCA)
  - Correspondence Analysis (CA)
  - Principal Coordinates Analysis (PCO or PCoA)
  - Non-metric Multidimensional Scaling (NMDS)
- Practical tips for working & plotting ordinations using **vegan**

Ordination

ordnung

Ordination

Putting things in order is exactly what we we do in ordination

- we arrange our samples along gradients by fitting lines and planes through the data that describe the main patterns in those data
- we map data to lower dimensions reflecting how similar the samples are to one another in terms of the variables measured

Three families of models

1. Linear
2. Unimodal
3. Distance-based

Unconstrained

Unconstrained

What is unconstrained?

First we look for major variation, then relate it to environmental variation

vs. *constrained* ordination, where we only want to see what can be explained by environmental variables of interest

How well do we explain the main patterns in the species data? vs *How large are the patterns we can expain with the measured data*?

Ordination methods

Principal Components Analysis (PCA) is a linear method — most useful for environmental data or sometimes with species data and short gradients

Correspondence Analysis (CA) is a unimodal method — most useful for species data, especially where non-linear responses are observed

Principal Coordinates Analysis (PCO) and Non-metric Multidimensional Scaling (NMDS) — can be used for any kind of data

PCA

Principal Components Analysis

Instead of doing many regressions, do one with all the responses

No explanatory variables — uncover latent, underlying gradients

PCA fits a line through our cloud of data in such a way that it maximises the variance in the data captured by that line (i.e.~minimises the distance between the fitted line and the observations)

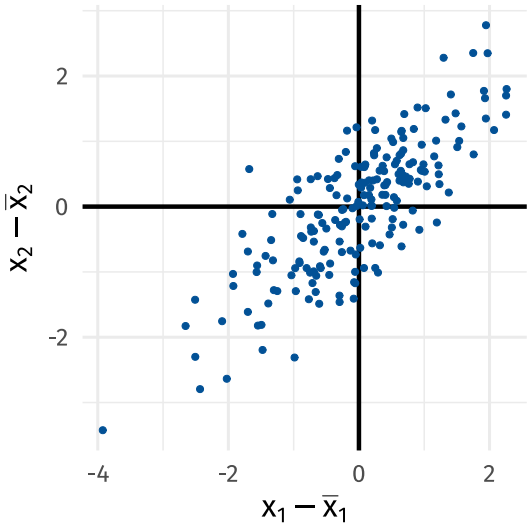
Fit a second line to form a plane, and so on, until we have one PCA axis for every dimension of the data

Each of these subsequent axes is uncorrelated with previous axes — they are orthogonal — the variance each axis explains is uncorrelated

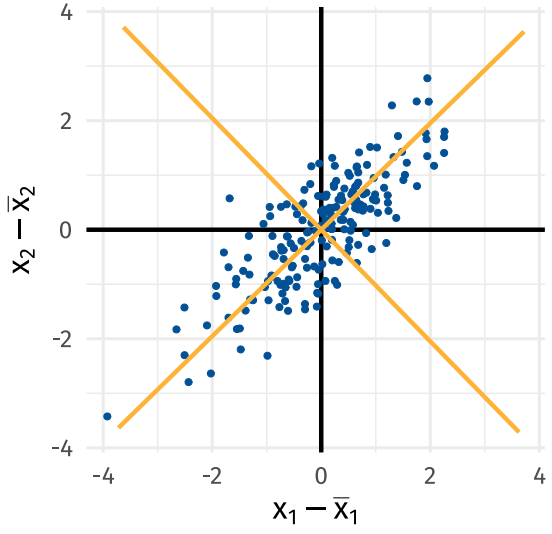
Principal Components Analysis

Principal Components Analysis

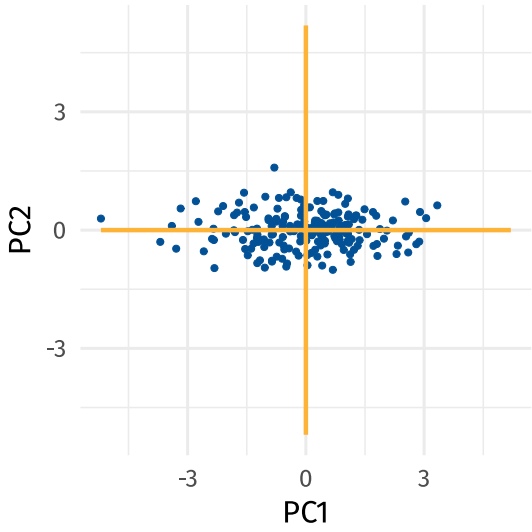
Centre variables



Find directions



Rotate



Principal Components Analysis

Load vegan

vegan is an add-on package

```
## install.packages("vegan") # Only need if you've never installed before
library("vegan")
data(varespec)
data(varechem)
```

vegan comes with a number of data sets which we'll use to get started

Vegetation in lichen pastures — species

```
class(varespec)
```

```
## [1] "data.frame"
```

```
dim(varespec)                                # number of samples, species
```

```
## [1] 24 44
```

```
head(varespec[,1:6], n = 5)
```

```
##    Callvulg Empenigr Rhodtome Vaccmyrt Vaccviti Pinusylv
## 18     0.55    11.13     0.00     0.00    17.80     0.07
## 15     0.67     0.17     0.00     0.35    12.13     0.12
## 24     0.10     1.55     0.00     0.00    13.47     0.25
## 27     0.00    15.13     2.42     5.92    15.97     0.00
## 23     0.00    12.68     0.00     0.00    23.73     0.03
```

`varespec` is a data frame

- *Variables* are the columns (here the species)
- *Observations* are the rows (the samples, sites, etc)

Vegetation in lichen pastures — chemistry

Also have associated soil physical and chemical measurements at the 24 sites

```
head(varechem)
```

```
##       N    P     K    Ca    Mg    S    Al   Fe    Mn   Zn  Mo Baresoil Humdepth
## 18 19.8 42.1 139.9 519.4  90.0 32.3  39.0 40.9  58.1  4.5 0.3     43.9       2.2
## 15 13.4 39.1 167.3 356.7  70.7 35.2  88.1 39.0  52.4  5.4 0.3     23.6       2.2
## 24 20.2 67.7 207.1 973.3 209.1 58.1 138.0 35.4  32.1 16.8 0.8     21.2       2.0
## 27 20.6 60.8 233.7 834.0 127.2 40.7  15.4  4.4 132.0 10.7 0.2     18.7       2.9
## 23 23.8 54.5 180.6 777.0 125.8 39.5  24.2  3.0  50.1  6.6 0.3     46.0       3.0
## 19 22.8 40.9 171.4 691.8 151.4 40.8 104.8 17.6  43.6  9.1 0.4     40.5       3.8
##     pH
## 18 2.7
## 15 2.8
## 24 3.0
## 27 2.8
## 23 2.7
## 19 2.7
```

Vegetation in lichen pastures — PCA

PCA is fitted using `rda()`

- Provide a data frame of observations on one or more variables
- To scale all variables to be $\mu = 0$, $\sigma^2 = 1$: `scale = TRUE`

```
pca ← rda(decostand(varespec, method = "hellinger"), scale = TRUE)
pca
```

```
## Call: rda(X = decostand(varespec, method = "hellinger"), scale = TRUE)
##
##               Inertia Rank
## Total              44
## Unconstrained      44   23
## Inertia is correlations
##
## Eigenvalues for unconstrained axes:
##   PC1   PC2   PC3   PC4   PC5   PC6   PC7   PC8
## 8.603 5.134 4.576 3.714 3.245 2.779 2.626 2.221
## (Showing 8 of 23 unconstrained eigenvalues)
```

Vegetation in lichen pastures — PCA

PCA of the covariance matrix — default is `scale = FALSE`

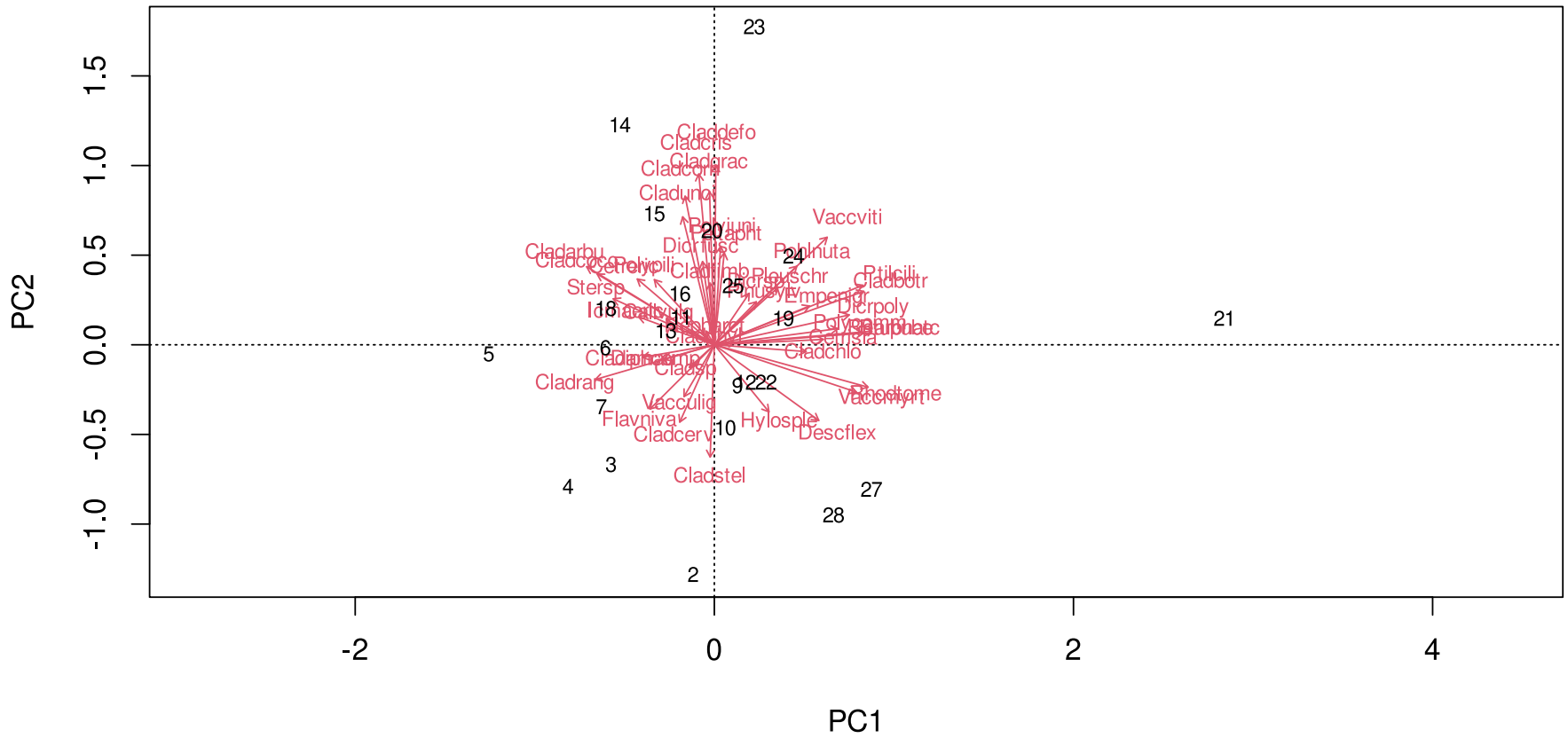
```
rda(decostand(varespec, method = "hellinger"), scale = FALSE)
```

```
## Call: rda(X = decostand(varespec, method = "hellinger"), scale = FALSE)
##
##               Inertia Rank
## Total          0.3647
## Unconstrained  0.3647   23
## Inertia is variance
##
## Eigenvalues for unconstrained axes:
##     PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8
## 0.14586 0.07908 0.02866 0.02446 0.02209 0.01263 0.01179 0.00873
## (Showing 8 of 23 unconstrained eigenvalues)
```

How **vegan** scales the eigenvalues is different to *Canoco*

Vegetation in lichen pastures — PCA

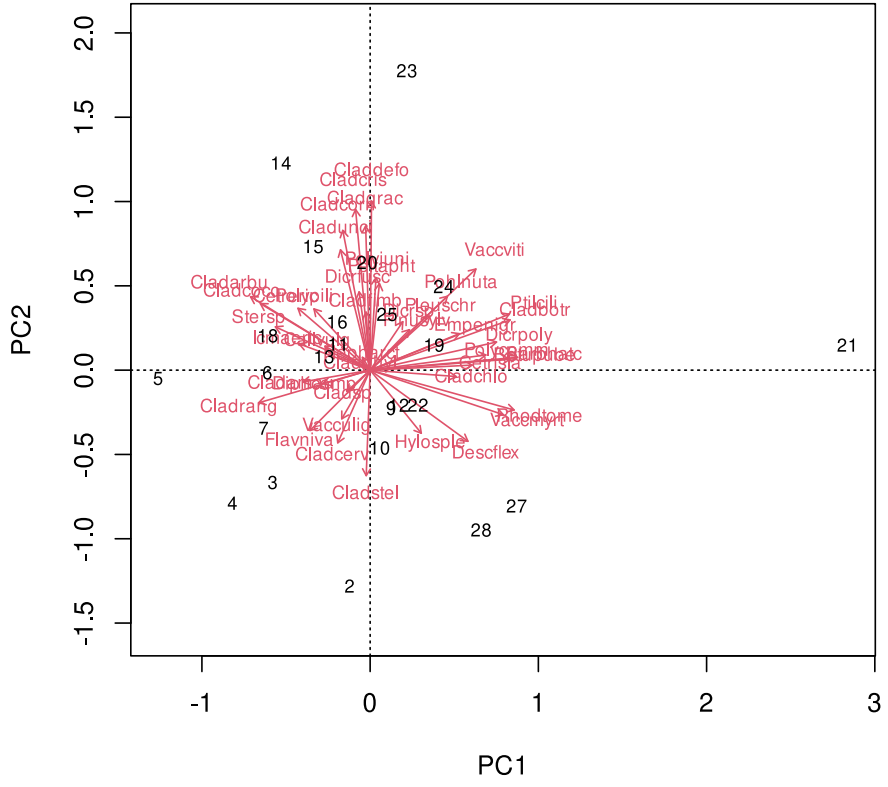
```
biplot(pca, scaling = "symmetric")
```



PCA biplots

- Sample (species) points plotted close together have similar species compositions (occur together)
- In PCA, species scores often drawn as arrows — point in direction of increasing abundance
- Species arrows with small angles to an axis are highly correlated with that axis

```
biplot(pca, scaling = "symmetric")
```



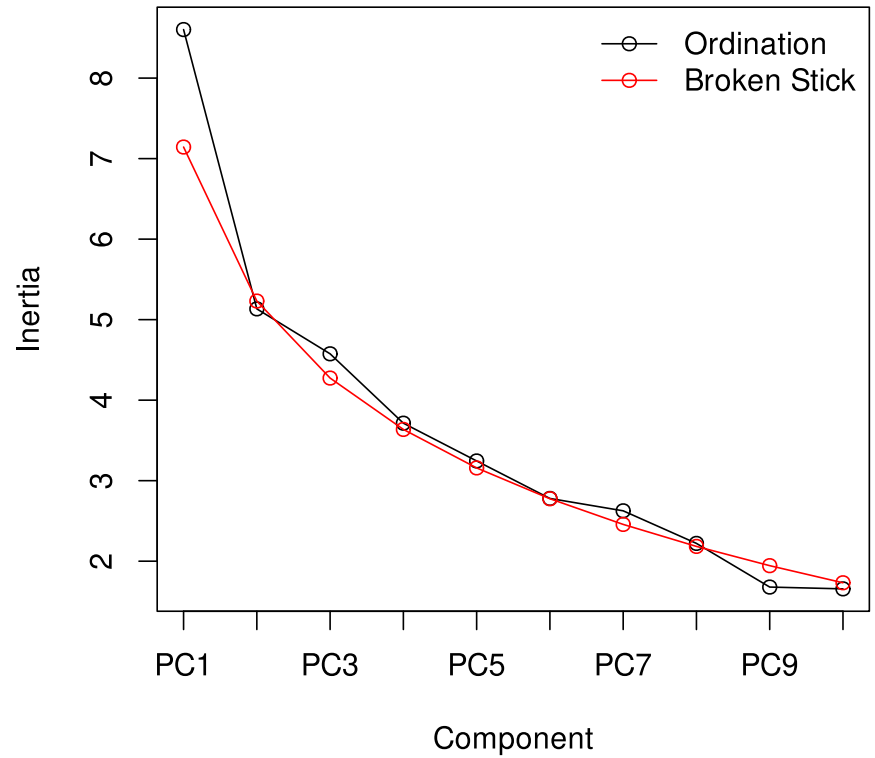
Eigenvalues λ

Eigenvalues are the amount of variance (inertia) explained by each axis

```
head(eigenvals(pca), 5)
```

```
##       PC1       PC2       PC3       PC4       PC5
## 8.602826 5.133623 4.575623 3.713926 3.244925
```

```
screeplot(pca, bstick = TRUE, type = "l",
          main = NULL)
```



Eigenvalues λ

The `summary()` method provides additional information

```
summary(eigenvals(pca))
```

```
## Importance of components:
##                            PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Eigenvalue              8.6028 5.1336 4.5756 3.71393 3.24492 2.77919 2.62560
## Proportion Explained    0.1955 0.1167 0.1040 0.08441 0.07375 0.06316 0.05967
## Cumulative Proportion   0.1955 0.3122 0.4162 0.50059 0.57434 0.63750 0.69718
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Eigenvalue              2.22100 1.67857 1.65627 1.30432 1.03472 0.91190 0.87323
## Proportion Explained    0.05048 0.03815 0.03764 0.02964 0.02352 0.02073 0.01985
## Cumulative Proportion   0.74765 0.78580 0.82344 0.85309 0.87660 0.89733 0.91718
##                           PC15   PC16    PC17    PC18     PC19     PC20     PC21
## Eigenvalue              0.7612 0.6336 0.52021 0.51641 0.408841 0.267175 0.206336
## Proportion Explained    0.0173 0.0144 0.01182 0.01174 0.009292 0.006072 0.004689
## Cumulative Proportion   0.9345 0.9489 0.96070 0.97243 0.981727 0.987799 0.992488
##                            PC22     PC23
## Eigenvalue              0.179715 0.150795
## Proportion Explained    0.004084 0.003427
## Cumulative Proportion   0.996573 1.000000
```

Correspondence Analysis

Correspondence Analysis

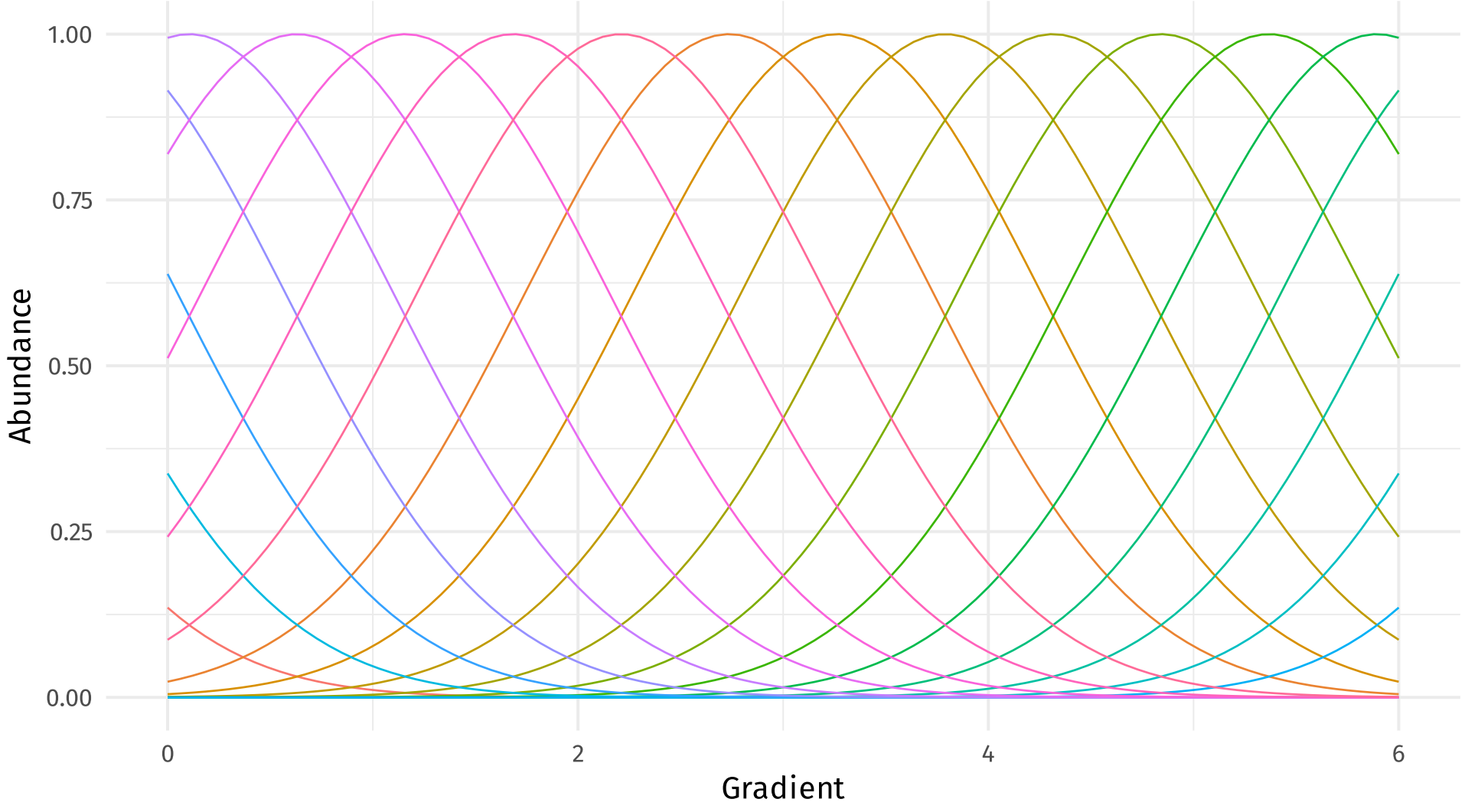
Correspondence analysis (CA) is very similar to PCA — a weighted form of PCA

The row and column sums are used as weights and this has the effect of turning the analysis into one of relative composition

The weighting is a trick to get linear-based software to fit non-linear responses

These nonlinear response are assumed to unimodal Gaussian curves, all with equal height and tolerance widths, and equally spaced optima

Correspondence Analysis



Correspondence Analysis

Correspondence analysis (CA) is very similar to PCA — a weighted form of PCA

The row and column sums are used as weights and this has the effect of turning the analysis into one of relative composition

The weighting is a trick to get linear-based software to fit non-linear responses

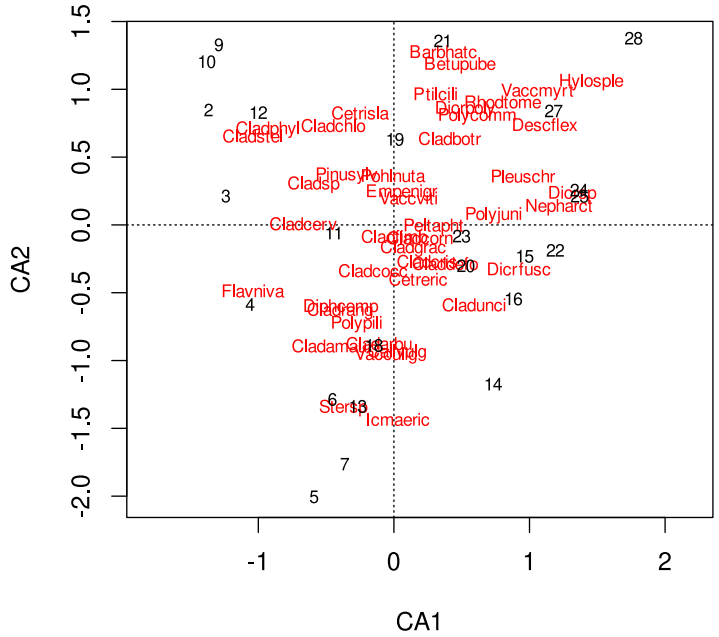
These nonlinear response are assumed to unimodal Gaussian curves, all with equal height and tolerance widths, and equally spaced optima

So, not very realistic, but it is surprisingly robust at times to violation of this assumption

Vegetation in lichen pastures — CA biplots

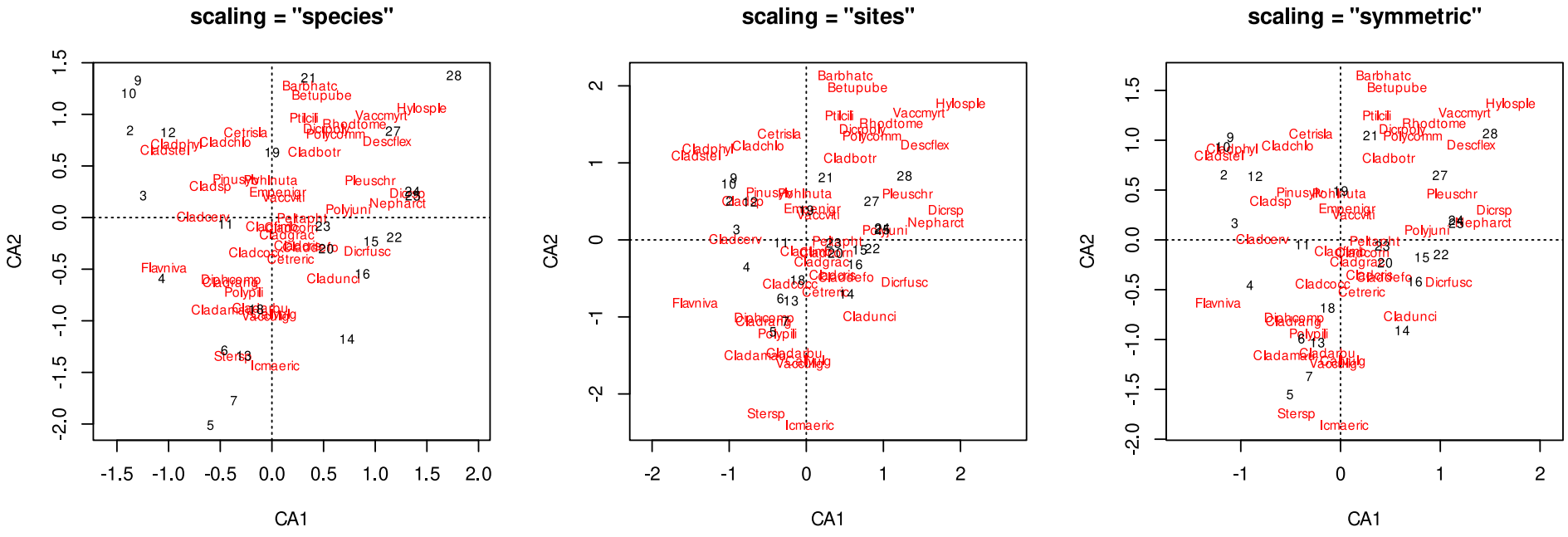
- Have two sets of scores
  1. Species scores
  2. Site scores
- Sample (species) points plotted close together have similar species compositions (occur together)
- In CA, species scores drawn as points — this is the fitted optima along the gradients
- Abundance of species declines in concentric circles away from the optima

```
ca ← cca(varespec)
plot(ca)
```



Vegetation in lichen pastures — CA biplots

- Species scores plotted as weighted averages of site scores, or
- Site scores plotted as weighted averages of species scores, or
- A symmetric plot



vegan

Vegan basics

- The majority of vegan functions work with a single vector, or more commonly an entire data frame
- This data frame may contain the species abundances
- Where subsidiary data is used/required, these two are supplied as data frames
- For example; the environmental constraints in a CCA
- It is not a problem if you have all your data in a single file/object; just subset it into two data frames after reading it into R

```
spp ← allMyData[, 1:20] ## columns 1-20 contain the species data
env ← allMyData[, 21:26] ## columns 21-26 contain the environmental data
```

`scores()` & `scaling`

- When we draw the results of many ordinations we display 2 or more sets of data
- Can't display all of these and maintain relationships between the scores
- Solution; scale one set of scores relative to the other
- Controlled via the `scaling` argument

Scaling

How we scale scores is controlled via the `scaling` argument

- `scaling = 1` — Focus on sites, scale site scores by λ_i
- `scaling = 2` — Focus on species, scale species scores by λ_i
- `scaling = 3` — Symmetric scaling, scale both scores by $\sqrt{\lambda_i}$
- `scaling = -1` — As above, but for `rda()` get correlation scores
- `scaling = -2` — for `cca()` multiply results by $\sqrt{(1/(1-\lambda_i))}$
- `scaling = -3` — this is Hill's scaling
- `scaling < 0` — For `rda()` divide species scores by species' σ
- `scaling = 0` — raw scores

No one can remember all that...

Use the text names instead:

- `scaling = "none"` means `scaling = 0`
- `scaling = "sites"` means `scaling = 1`
- `scaling = "species"` means `scaling = 3`
- `scaling = "symmetric"` means `scaling = 3`

For PCA (`rda()`) use `correlation = TRUE` to get the correlation scores (negative scaling)

For CA (`cca()`) use `hill = TRUE` to get Hill's scaling scores (negative scaling)

Extractor functions — scores()

Don't rummage around in the objects returned by *vegan* functions — unless you know what you're doing

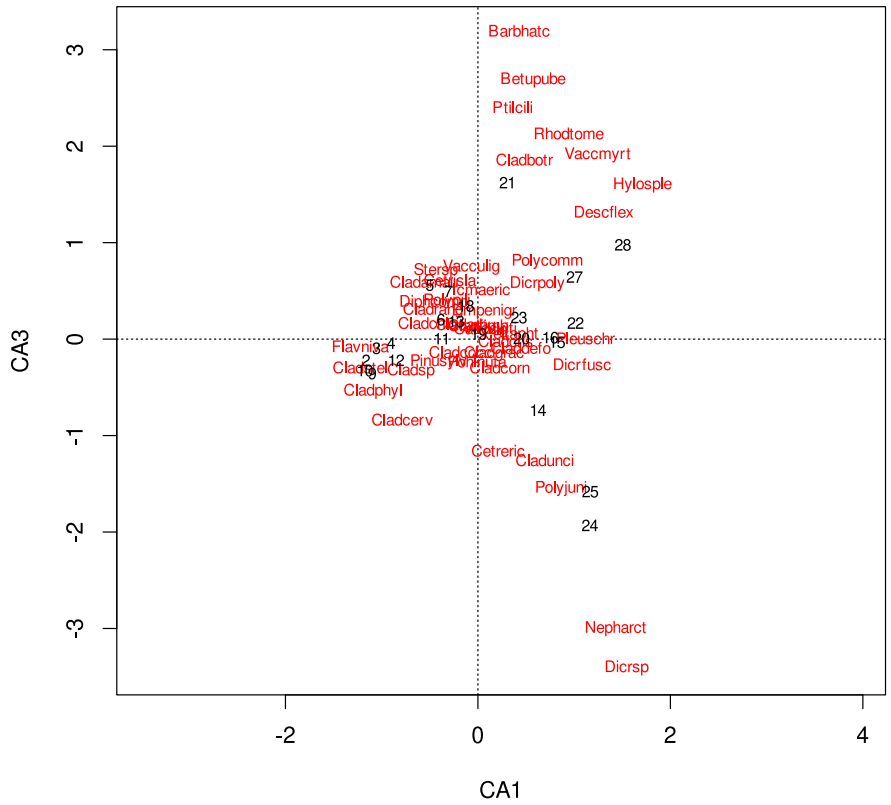
```
head(scores(pca, choices = 1:3, display = "species", scaling = "species", correlation = TRUE))
```

```
##                 PC1        PC2         PC3
## Callvulg -0.2896130  0.1338259  0.18460432
## Empenigr  0.5732742  0.2057975 -0.24778607
## Rhodtome  0.9262883 -0.2248872 -0.28211297
## Vaccmyrt  0.8558251 -0.2508985 -0.33438598
## Vaccviti  0.6800662  0.5703573  0.02059066
## Pinusylv  0.2530739  0.2280044  0.80172865
```

Basic ordination plots — plot()

- `choices = 1:2` — which axes?
- `scaling = 3` — scaling to use
- `display = c("sites","species")` — which scores (default is both)
- `type = "text"` — display scores using labels or points (`"points"`)
- Other graphics arguments can be supplied but apply to all scores

```
plot(ca1, choices = c(1,3),
     scaling = "symmetric")
```



Distance-based methods

PCO

Principal Coordinates Analysis

PCoA (or PCO, or **metric** multidimensional scaling (MDS)) finds a mapping to Euclidean space of n objects using the n by n matrix of dissimilarities d_{ij}

PCoA is an eigen decomposition like PCA

- first axis is the best 1D mapping of the dissimilarities
- subsequent axes are orthogonal to the first, but improve the mapping by smaller & smaller amounts

Can use *any* dissimilarity coefficient (with a big **but**)

PCoA on a Euclidean distance matrix ⇨ PCA (without species scores)

Principal Coordinates Analysis

The big **but** is that not all dissimilarity coefficients can be represented in Euclidean space

If dissimilarity matrix is metric we're OK — *usually*

If not metric, get negative eigenvalues ⇨ correspond to distances in imaginary space

Distortion can be measured as

$$\frac{\sum |\lambda^{-}|}{\sum |\lambda|}$$

PCoA — correcting negative λ

1. Could square root transform the d_{ij}
2. Add a sufficiently large constant to d_{ij} or d_{ij}^2
   - **Lingoes** method: $\hat{d}_{ij} = \sqrt{d_{ij}^2 + 2c_1}$ for $i \neq j$

     where c_1 is max $\left(\left|\lambda_i^-\right|\right)$
   - **Cailliez** method: $\hat{d}_{ij} = d_{ij} + c_2$ for $i \neq j$

     where c_2 is computed from a special matrix formed during the PCoA calculations

Principal Coordinates Analysis

Default dissimilarity in `vegdist()` is Bray-Curtis

```
pco1 ← wcmdscale(vegdist(varespec), eig = TRUE)
round(eigenvals(pco1), 3)
```

```
##  [1]  1.755  1.133  0.443  0.370  0.245  0.196  0.175  0.128  0.097  0.076
## [11]  0.064  0.058  0.039  0.017  0.005  0.000 -0.006 -0.013 -0.025 -0.038
## [21] -0.048 -0.054 -0.074
```

```
pco2 ← wcmdscale(vegdist(varespec), eig = TRUE, add = "lingoes")
round(eigenvals(pco2), 3)
```

```
##  [1] 1.829 1.208 0.517 0.444 0.319 0.270 0.249 0.203 0.171 0.150 0.138 0.132
## [13] 0.114 0.091 0.079 0.074 0.068 0.061 0.049 0.037 0.026 0.020
```

Note there's one fewer dimensions after correction

Principal Coordinates Analysis

Can plot the PCoA using `plot()` and add *species scores*

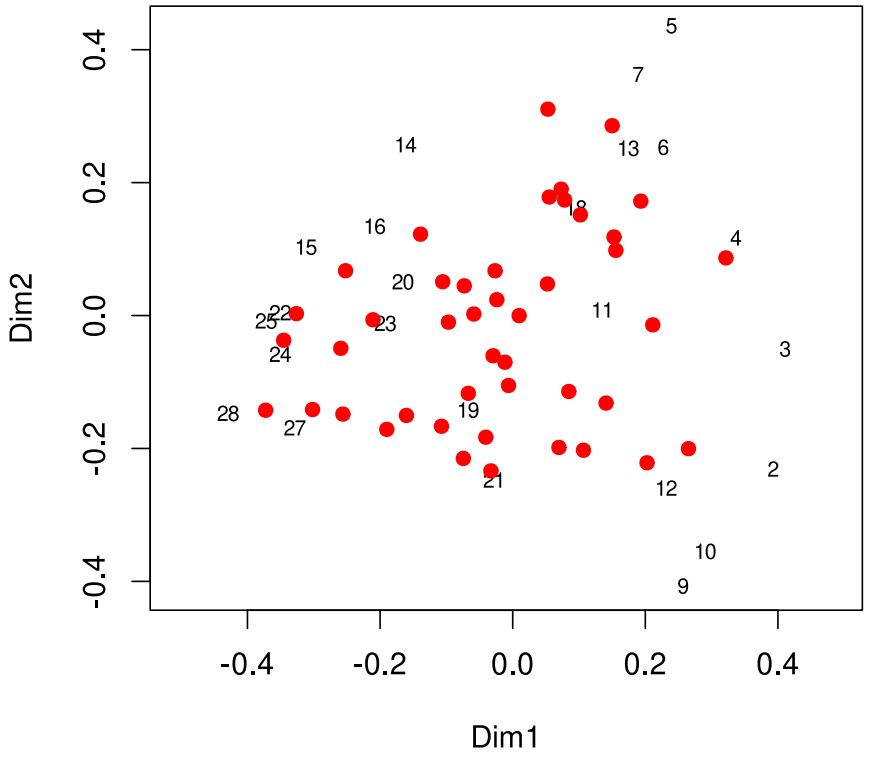
```
pco ← wcmdscale(vegdist(varespec), eig = TRUE)

## plot
plot(pco)

## get PCoA scores
scrs ← scores(pco, choices = 1:2)

## take WA of PCoA scores,
## weight by abundance
spp_scrs ← wascores(scrs, varespec,
                    expand = FALSE)

## add
points(spp_scrs, col = "red", pch = 19)
```



NMDS

Non-Metric Multidimensional Scaling

NMDS find a low-dimensional mapping that preserves as best as possible the rank order of the original dissimilarities d_{ij}

Solution with minimal `stress` is sought; a measure of how well the NMDS mapping fits the d_{ij}

Stress is sum of squared residuals of monotonic regression between distances in NMDS space, d^*_{ij}, & d_{ij}

Non-linear regression can cope with non-linear responses in species data

Iterative solution; *convergence is not guaranteed*

Must solve separately different dimensionality solutions

Non-Metric Multidimensional Scaling

- Use an appropriate dissimilarity metric that gives good gradient separation `rankindex()`, e.g Bray-Curtis, Jaccard, Kulczynski
- Wisconsin transformation useful; Standardize species to equal maxima, then sites to equal totals `wisconsin()`
- Use many random starts and look at the fits with lowest stress (`try` & `trymax`
- Only conclude solution reached if lowest stress solutions are similar (Procrustes rotation)
- Fit NMDS for 1, 2, 3, ... dimensions; stop after a sudden drop in stress observed in a screeplot
- NMDS solutions can be rotated ; common to rotate to PCs
- Also scale axes in half-change units

NMDS in vegan

vegan implements all these ideas via the `metaMDS()` wrapper

```
data(dune)
set.seed(10)
(sol ← metaMDS(dune, trace = FALSE))
```

```
##
## Call:
## metaMDS(comm = dune, trace = FALSE)
##
## global Multidimensional Scaling using monoMDS
##
## Data:     dune
## Distance: bray
##
## Dimensions: 2
## Stress:     0.1183186
## Stress type 1, weak ties
## Best solution was repeated 6 times in 20 tries
## The best solution was from try 2 (random start)
## Scaling: centring, PC rotation, halfchange scaling
## Species: expanded scores based on 'dune'
```

NMDS in vegan

If no convergent solutions, continue iterations from previous best solution

```
(sol ← metaMDS(dune, previous.best = sol, trace = FALSE))
```

```
##
## Call:
## metaMDS(comm = dune, trace = FALSE, previous.best = sol)
##
## global Multidimensional Scaling using monoMDS
##
## Data:     dune
## Distance: bray
##
## Dimensions: 2
## Stress:     0.1183186
## Stress type 1, weak ties
## Best solution was repeated 16 times in 40 tries
## The best solution was from try 2 (random start)
## Scaling: centring, PC rotation, halfchange scaling
## Species: expanded scores based on 'dune'
```

NMDS in vegan

```
layout(matrix(1:2, ncol = 2))
plot(sol, main = "Dune NMDS plot"); stressplot(sol, main = "Shepard plot")
layout(1)
```

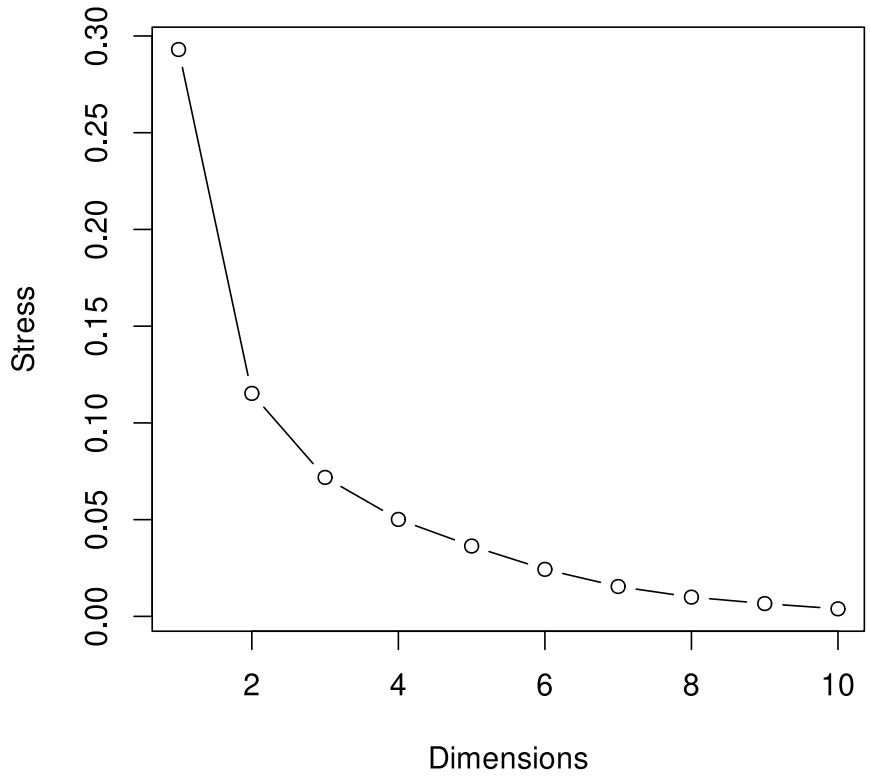
Checking dimensionality *k*

Fit NMDS solutions for a number of k

```
k_vec ← 1:10
stress ← numeric(length(k_vec))
dune_dij ← metaMDSdist(dune, trace = FALSE)
set.seed(25)
for(i in seq_along(k_vec)) {
    sol ← metaMDSiter(dune_dij, k = i,
                      trace = FALSE)
    stress[i] ← sol$stress
}
```

Need to use the helper functions to do this right

```
plot(k_vec, stress, type = "b", ylab =
"Stress",
     xlab = "Dimensions")
```



NMDS — Goodness of fit

A goodness of fit statistic g_i can be computed for the observations; defined such that $\sum_{i=1}^{n} g_i^2 = S^2$

```
(g ← goodness(sol))
```

```
##  [1] 0.0008413322 0.0014436954 0.0005931447 0.0010673136 0.0008359822
##  [6] 0.0006836062 0.0010013178 0.0007222658 0.0007036269 0.0009243645
## [11] 0.0003849299 0.0005012098 0.0008889085 0.0011565531 0.0013128102
## [16] 0.0006928218 0.0008133972 0.0005597081 0.0008862855 0.0006112743
```

```
sum(g^2)
```

```
## [1] 1.518778e-05
```

```
sol$stress^2
```

```
## [1] 1.518778e-05
```

```
all.equal(sqrt(sum(g^2)), sol$stress)
```

```
## [1] TRUE
```

Supplementary data

If we have other data collected at the same sites, say about the environment, we can investigate relationships between the main components of variation in species composition and those environmental variables

Two main **vegan** functions for this

1. `envfit()` and helpers `vectorfit()` and `factorfit()`
2. `ordisurf()`

`envfit()` fits vectors or planes while `ordisurf()` fits smooth, potentially non-linear surfaces

Vector fitting

`envfit()` fits vectors using a regression

$$\hat{y}_i = \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots \beta_p x_{pi}$$

where:

- x_{pi} are the site scores for the i^{th} site on axis 1, 2, ..., p
- Usually $p = 2$ as we're fitting vectors into 2D ordination plots
- y_i is the value of the environment at the i^{th} site
- `envfit()` can handle a matrix of environmental variables

Note no intercept; internally center $\boldsymbol{y}$ and $\boldsymbol{x_p}$

For categorical $\boldsymbol{y}$, `envfit()` finds averages of scores for each level of the category

Vector fitting

`envfit()` with formula

```
set.seed(42)
ev ← envfit(pca ~ ., data = varechem,
            choices = 1:2,
            scaling = "symmetric",
            permutations = 1000)
ev
```

or data frames

```
envfit(pca, varechem,
       choices = 1:2, scaling = "symmetric",
       permutations = 1000)
```

```
##
## ***VECTORS
##
##                PC1       PC2      r2   Pr(>r)
## N         -0.80597 -0.59196 0.0384 0.676324
## P          0.78221 -0.62301 0.0335 0.696304
## K          0.94909 -0.31501 0.0394 0.638362
## Ca         0.87606  0.48220 0.1607 0.153846
## Mg         0.83079  0.55658 0.1774 0.121878
## S          0.86667 -0.49888 0.0044 0.962038
## Al        -0.64541 -0.76384 0.3168 0.017982 *
## Fe        -0.56956 -0.82195 0.3536 0.018981 *
## Mn         0.96802 -0.25088 0.0730 0.439560
## Zn         0.96899  0.24711 0.0321 0.712288
## Mo        -0.99233 -0.12362 0.0630 0.491508
## Baresoil   0.51078  0.85971 0.5280 0.000999
***
## Humdepth   0.81116  0.58483 0.2937 0.026973 *
## pH        -0.34311 -0.93929 0.0881 0.333666
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*'
0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 1000
```

Vector fitting

Values in PC1 etc are *direction cosines* for vectors of unit length

r2 is the squared correlation, R^2, between $\boldsymbol{y}$ and projection of axis scores on to the vector

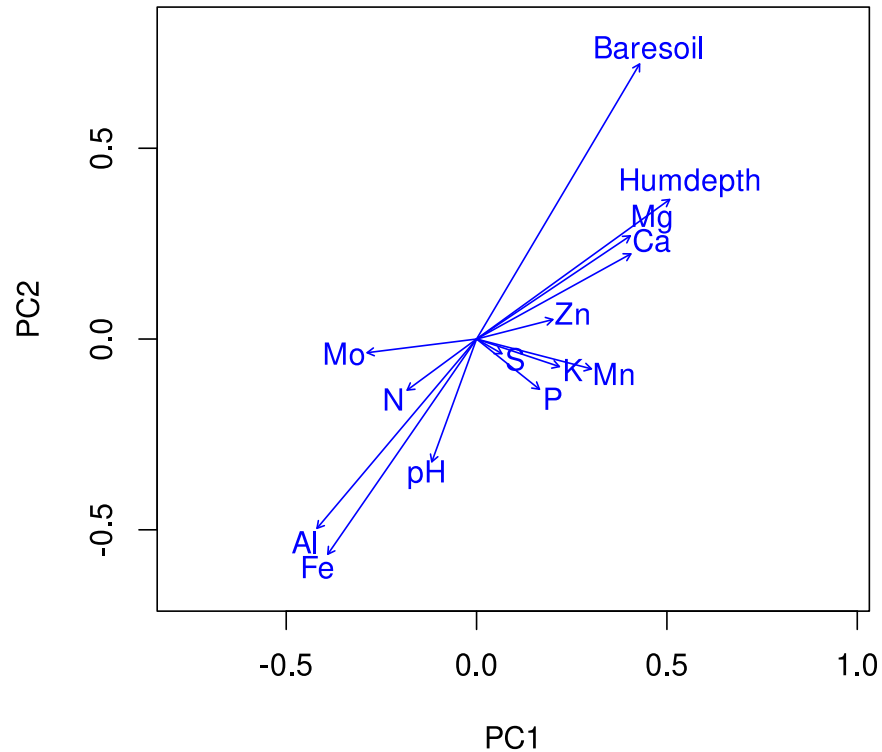
For factors is $R^2 = 1 - \mathrm{SS}_\mathrm{W}/\mathrm{SS}_\mathrm{T}$

Permutation test shuffles the $\boldsymbol{y}$ to generate distribution of R^2 under H_0

```
##
## ***VECTORS
##
##                PC1       PC2      r2   Pr(>r)
## N          -0.80597 -0.59196 0.0384 0.676324
## P           0.78221 -0.62301 0.0335 0.696304
## K           0.94909 -0.31501 0.0394 0.638362
## Ca          0.87606  0.48220 0.1607 0.153846
## Mg          0.83079  0.55658 0.1774 0.121878
## S           0.86667 -0.49888 0.0044 0.962038
## Al         -0.64541 -0.76384 0.3168 0.017982 *
## Fe         -0.56956 -0.82195 0.3536 0.018981 *
## Mn          0.96802 -0.25088 0.0730 0.439560
## Zn          0.96899  0.24711 0.0321 0.712288
## Mo         -0.99233 -0.12362 0.0630 0.491508
## Baresoil    0.51078  0.85971 0.5280 0.000999
***
## Humdepth    0.81116  0.58483 0.2937 0.026973 *
## pH         -0.34311 -0.93929 0.0881 0.333666
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*'
0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 1000
```

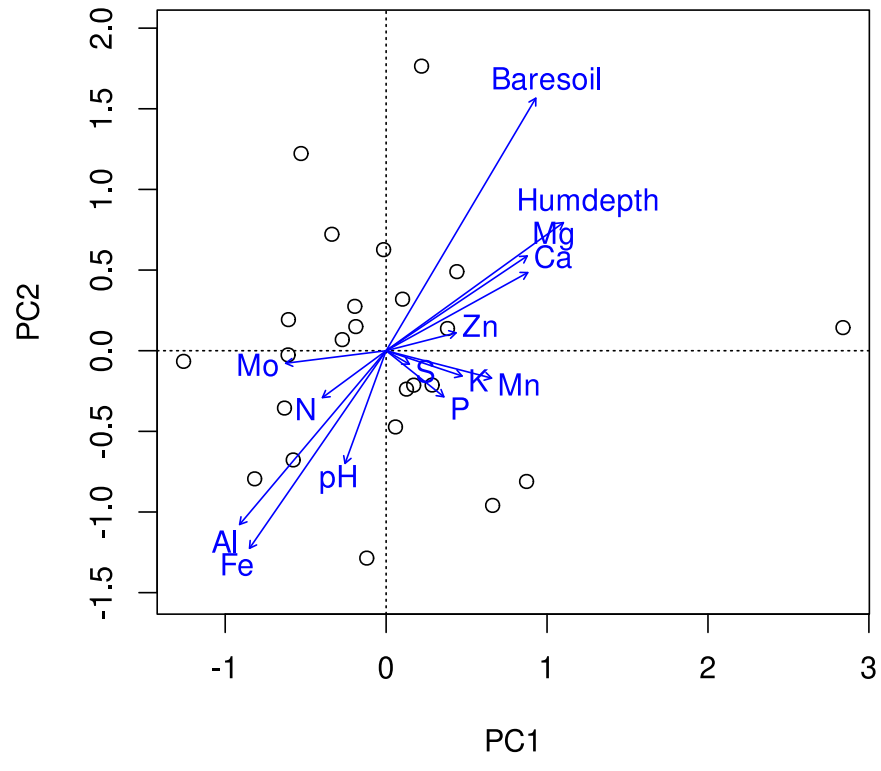
Vector fitting

```
plot(ev, add = FALSE) # oops bug!
```



Vector fitting

```
plot(pca, display = "sites", type = "n",
     scaling = "symmetric")
points(pca, display = "sites",
       scaling = "symmetric")
plot(ev, add = TRUE)
```



Smooth surfaces

`envfit()` fitted vectors, linear planes, to ordinations.

`ordisurf()` fits smooth surfaces using a GAM via package **mgcv**

$$\hat{y}_i = f(x_{1i}, x_{2i})$$

where $f()$ is a *bivariate* smooth function of a pair of axis scores x_{1i} and x_{2i}

`ordisurf()` exposes a lot of functionality from `mgcv::gam()` and the smooths it can fit

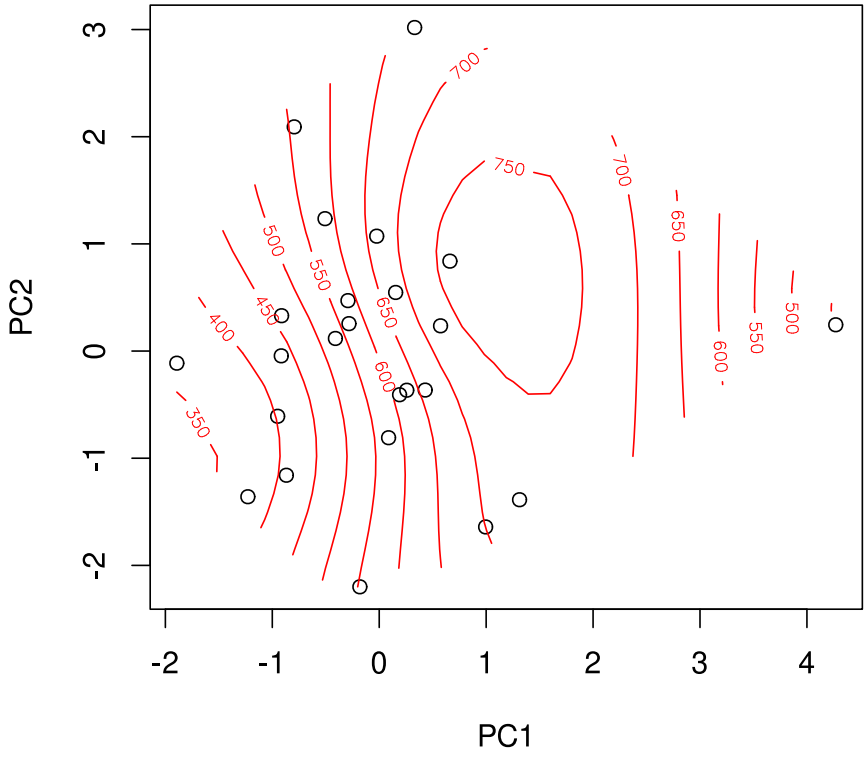
Smooth surfaces

Fitting a 10 basis function isotropic surface

```
surf ← ordisurf(pca ~ Ca,
                data = varechem,
                knots = 10,
                isotropic = TRUE,
                main = NULL)
```

`ordisurf()` plots by default

`surf` contains the fitted `gam()` model



Smooth surfaces

```
summary(surf)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, x2, k = 10, bs = "tp", fx = FALSE)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   569.66      40.27   14.14 1.05e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##            edf Ref.df    F p-value
## s(x1,x2) 3.517      9 1.34  0.0187 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.344   Deviance explained = 44.4%
## -REML = 158.56  Scale est. = 38924     n = 24
```

plotting

Better plotting

Ordination diagrams are often messy — `plot()` methods designed to get a quick plot of results

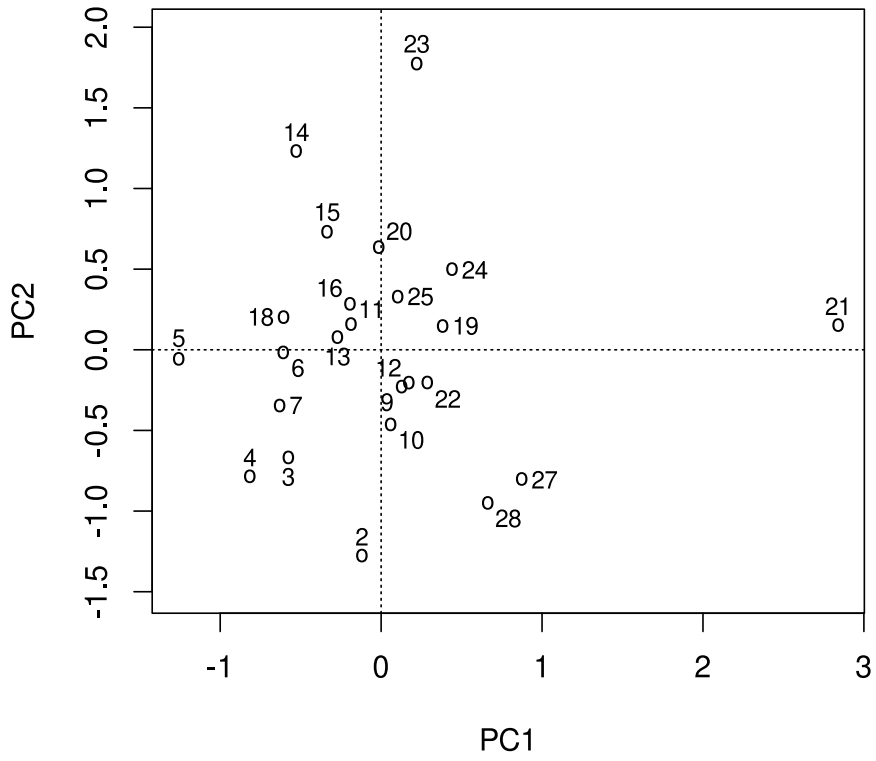
To produce better plots you need to know some base graphics skills *and* make use of some *vegan* helpers

Points and labels

`ordipointlabel()` can draw points and labels

```
set.seed(10)
ordipointlabel(pca,
               display = "sites",
               scaling = "symmetric")
```

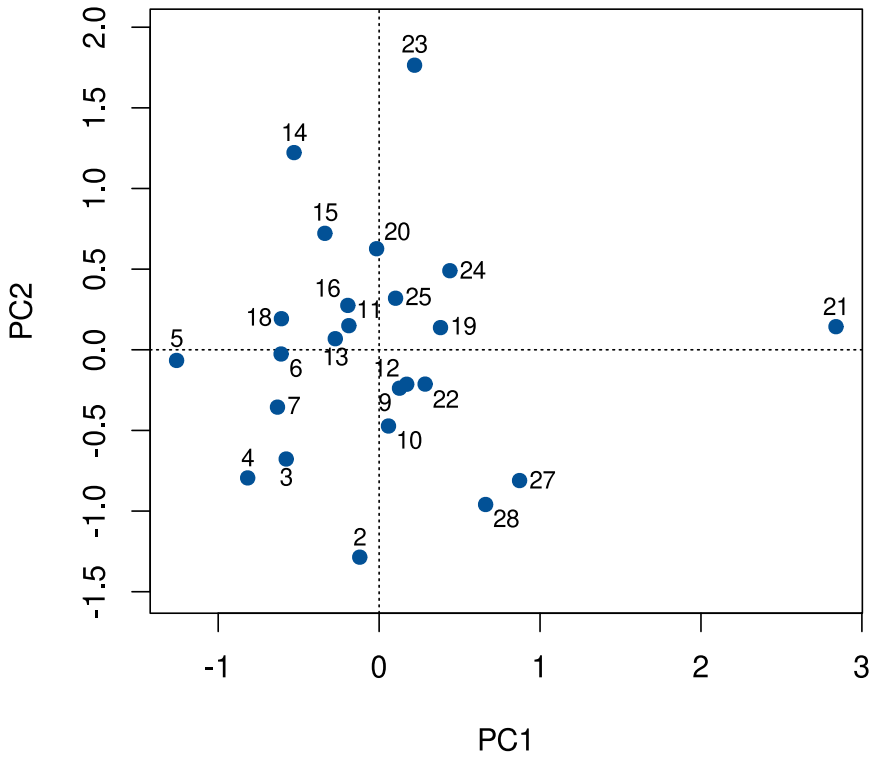
Iteratively finds space to draw labels so they don't overplot



Points and labels

`ordipointlabel()` can also add to an existing plot

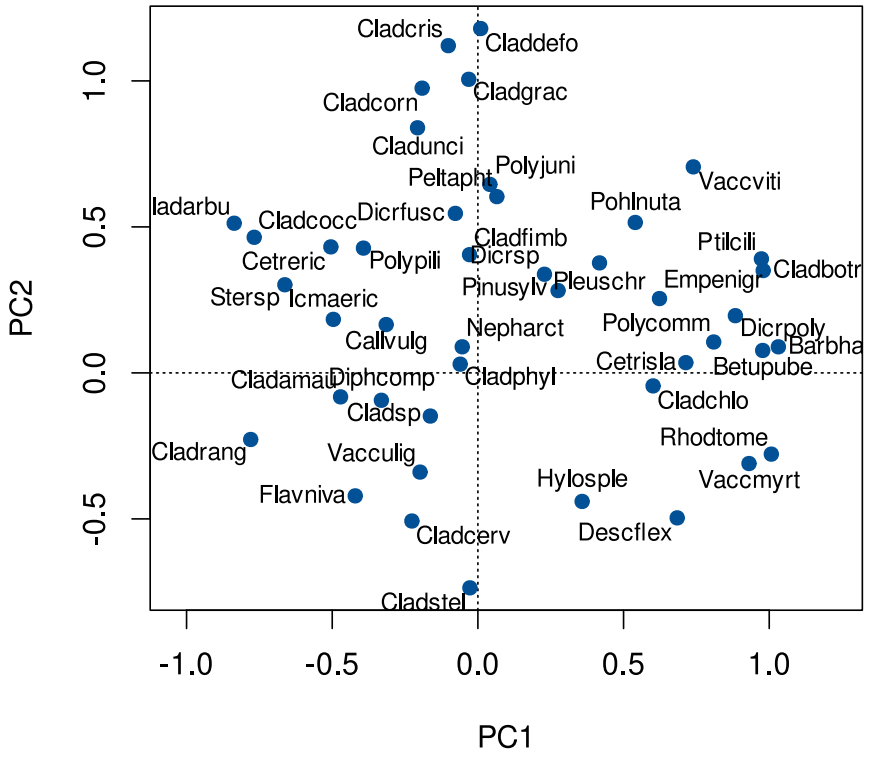
```
plot(pca, display = "sites",
     scaling = "symmetric", type = "n")
points(pca, display = "sites",
       scaling = "symmetric", pch = 19,
       col = "#025196")
set.seed(10)
ordipointlabel(pca,
               display = "sites",
               scaling = "symmetric",
               add = TRUE)
```



Points and labels

`ordipointlabel()` can also add to an existing plot

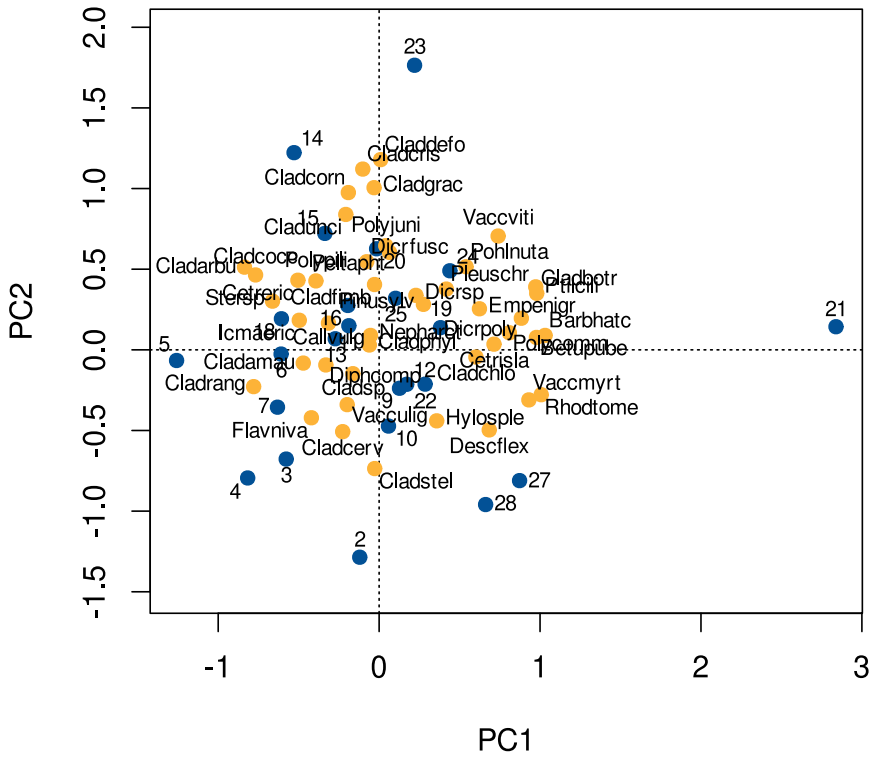
```
disp ← "species"
scl ← "symmetric"
plot(pca, display = disp,
     scaling = scl, type = "n")
points(pca, display = disp,
       scaling = scl, pch = 19,
       col = "#025196")
set.seed(10)
ordipointlabel(pca,
               display = disp,
               scaling = scl,
               add = TRUE)
```



Points and labels

How successful `ordipointlabel()` is depends on how much you plot & how big you plot it

```
disp ← c("sites", "species")
scl ← "symmetric"
plot(pca, display = disp,
     scaling = scl, type = "n")
points(pca, display = disp[1],
       scaling = scl, pch = 19,
       col = "#025196")
points(pca, display = disp[2],
       scaling = scl, pch = 19,
       col = "#fdb338")
set.seed(10)
ordipointlabel(pca,
               display = disp,
               scaling = scl,
               add = TRUE,
               col = c(1,1), cex = c(0.7, 0.7))
```



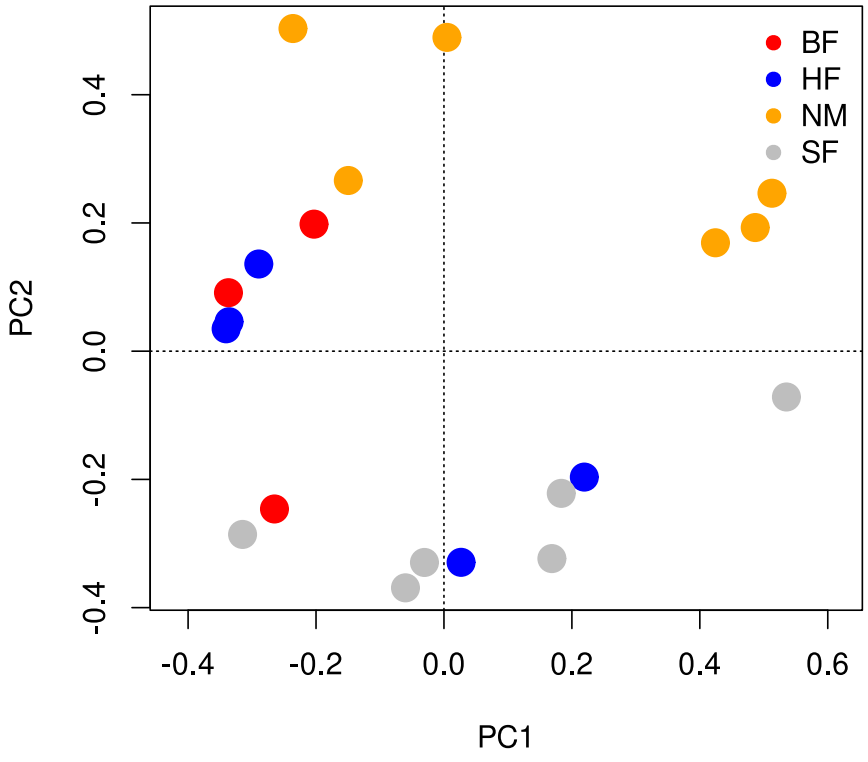
Building up by layers

With base graphics you are in control of *everything*

```
data(dune, dune.env)
col_vec ← c("red", "blue", "orange", "grey")
disp ← "sites"
scl ← "symmetric"

ord ← rda(decostand(dune, method="hellinger"))

plot(ord, type = "n", scaling = scl,
     display = disp)
cols ← with(dune.env, col_vec[Management])
points(ord, display = disp, scaling = scl,
       pch = 19, col = cols, cex = 2)
lvl ← with(dune.env, levels(Management))
legend("topright", legend = lvl,
       bty = "n", col = col_vec, pch = 19)
```

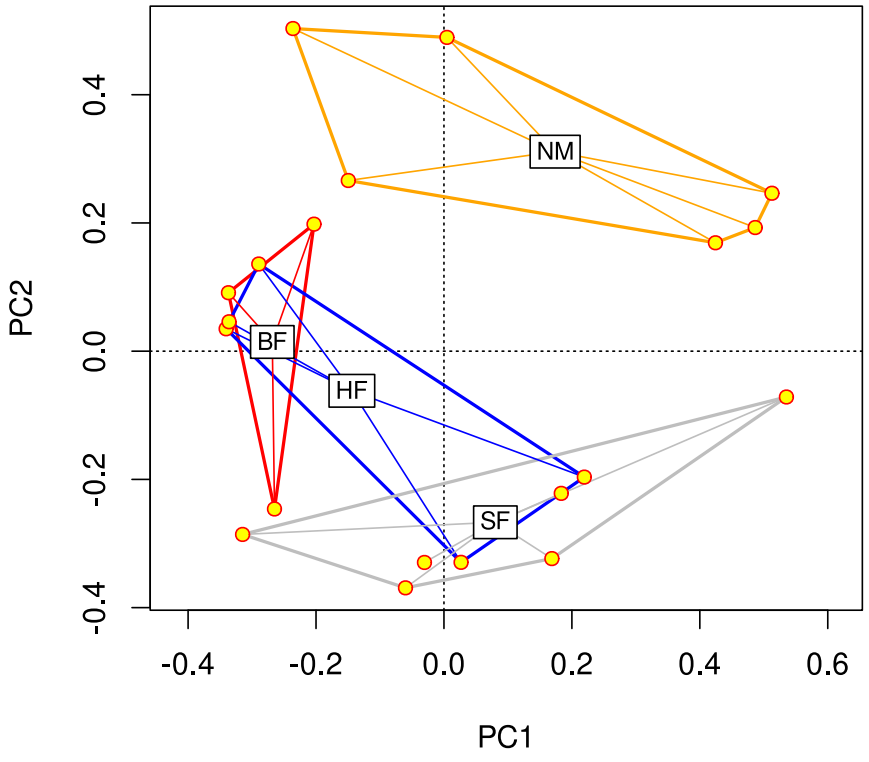


Other utilities — `ordihull()`

Convex hulls around groups of data

```
disp ← "sites"
scl ← "symmetric"

plot(ord, type = "n", scaling = scl,
     display = disp)
ordihull(ord, groups = dune.env$Management,
         col = col_vec,
         scaling = scl, lwd = 2)
ordispider(ord, groups = dune.env$Management,
           col = col_vec,
           scaling = scl, label = TRUE)
points(ord, display = disp, scaling = scl,
       pch = 21, col = "red", bg = "yellow")
```

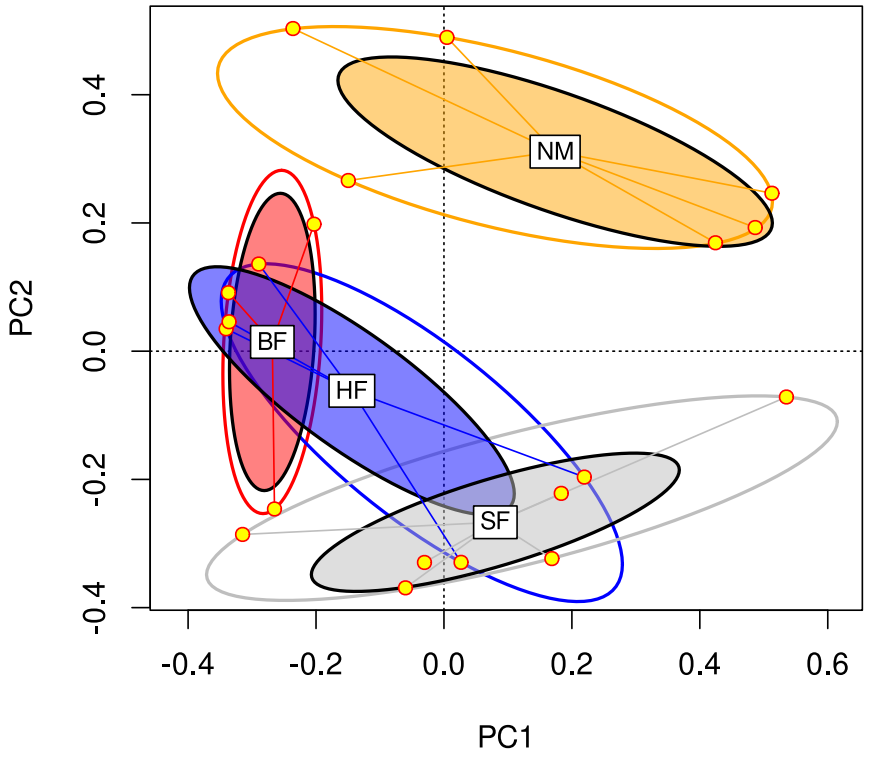


Other utilities — `ordiellipse()`

Draws ellipsoid hulls & standard error & deviation ellipses

```
disp ← "sites"
scl ← "symmetric"

plot(ord, type = "n", scaling = scl,
     display = disp)
## ellipsoid hull
ordiellipse(ord, groups = dune.env$Management,
            kind = "ehull", col = col_vec,
            scaling = scl, lwd = 2)
## standard error of centroid  ellipse
ordiellipse(ord, groups = dune.env$Management,
            draw = "polygon", col = col_vec,
            scaling = scl, lwd = 2)
ordispider(ord, groups = dune.env$Management,
           col = col_vec,
           scaling = scl, label = TRUE)
points(ord, display = disp, scaling = scl,
       pch = 21, col = "red", bg = "yellow")
```

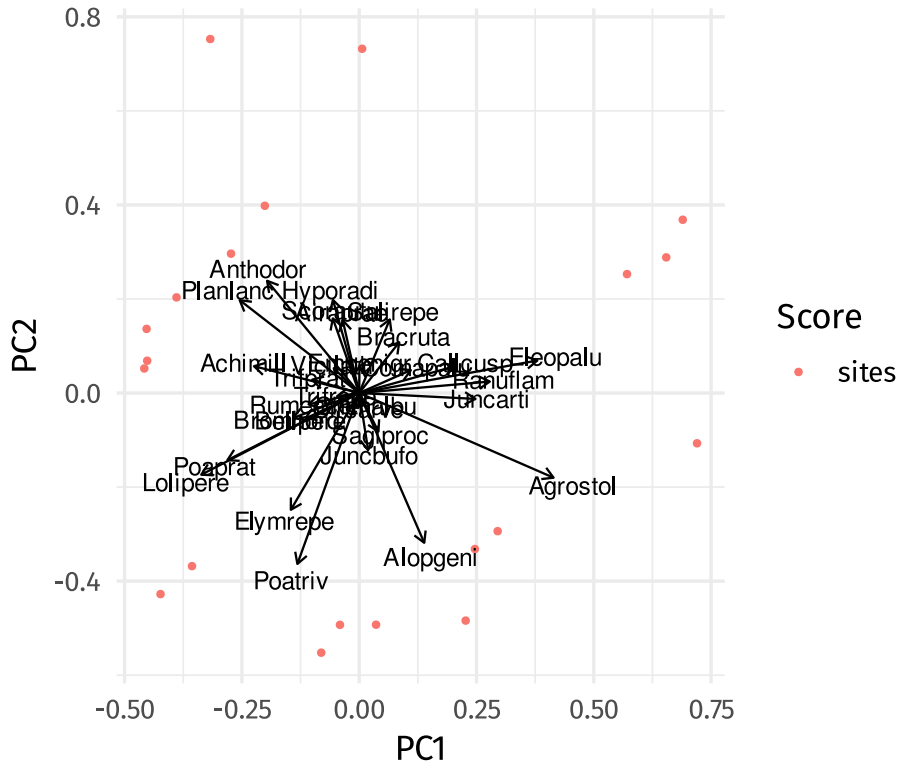


Any love for *ggplot*?

The **ggvegan** package is in development

```
## library('remotes')
## install_github("gavinsimpson/ggvegan")
library('ggvegan')
library('ggplot2')

autoplot(ord)
```



Any love for *ggplot*?

The **ggvegan** package is in development

```
# Install src from github
remotes::install_github("gavinsimpson/ggvegan")
# or binary from R-universe:
# Enable repository from gavinsimpson
options(repos = c(
  gavinsimpson = 'https://gavinsimpson.r-universe.dev',
  CRAN = 'https://cloud.r-project.org'))
# Download and install ggvegan in R
install.packages('ggvegan')
```

ggvegan

```
ford ← fortify(ord, axes = 1:2)
## not yet a tibble
head(ford, 4)
```

```
##      Score    Label          PC1          PC2
## 1 species Achimill -0.22413552  0.05704791
## 2 species Agrostol  0.41424457 -0.18006829
## 3 species Airaprae -0.03506944  0.15464367
## 4 species Alopgeni  0.13865500 -0.31844362
```

```
blue ← "#025196"
filter(ford, Score == "sites") %>%
    ggplot(aes(x = PC1, y = PC2)) +
    geom_hline(yintercept = 0, colour = blue) +
    geom_vline(xintercept = 0, colour = blue) +
    geom_point() +
    coord_fixed()
```