Constrained ordination

Gavin L. Simpson

February 12, 2025

Welcome

Today's topics

- Constrained ordination
  - Canonical Correspondence Analysis
  - Redundancy Analysis
  - Partial constrained ordination
- Model Building
  - Model selection
- Permutation tests

Constrained Ordination

CCA

Canonical Correspondence Analysis

CCA is the constrained form of CA; fitted using `cca()`

Two interfaces for specifying models

- basic; `cca1 ← cca(X = varespec, Y = varechem)`
- formula; `cca1 ← cca(varespec ~ ., data = varechem)`

RDA is the constrained form of PCA; fitted using `rda()`

Formula interface is the more powerful — *recommended*

Canonical Correspondence Analysis

```
cca1 ← cca(varespec ~ ., data = varechem)
cca1
```

```
## Call: cca(formula = varespec ~ N + P + K + Ca + Mg + S + Al + Fe + Mn +
## Zn + Mo + Baresoil + Humdepth + pH, data = varechem)
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Constrained    1.4415     0.6920   14
## Unconstrained  0.6417     0.3080    9
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##   CCA1   CCA2   CCA3   CCA4   CCA5   CCA6   CCA7   CCA8   CCA9  CCA10  CCA11
## 0.4389 0.2918 0.1628 0.1421 0.1180 0.0890 0.0703 0.0584 0.0311 0.0133 0.0084
##  CCA12  CCA13  CCA14
## 0.0065 0.0062 0.0047
##
## Eigenvalues for unconstrained axes:
##     CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8     CA9
## 0.19776 0.14193 0.10117 0.07079 0.05330 0.03330 0.01887 0.01510 0.00949
```

Redundancy Analysis

```
rda1 ← rda(varespec ~ ., data = varechem)
rda1
```

```
## Call: rda(formula = varespec ~ N + P + K + Ca + Mg + S + Al + Fe + Mn +
## Zn + Mo + Baresoil + Humdepth + pH, data = varechem)
##
##                Inertia Proportion Rank
## Total         1825.6594     1.0000
## Constrained   1459.8891     0.7997   14
## Unconstrained  365.7704     0.2003    9
## Inertia is variance
##
## Eigenvalues for constrained axes:
##  RDA1  RDA2  RDA3  RDA4  RDA5  RDA6  RDA7  RDA8  RDA9 RDA10 RDA11 RDA12 RDA13
## 820.1 399.3 102.6  47.6  26.8  24.0  19.1  10.2   4.4   2.3   1.5   0.9   0.7
## RDA14
##   0.3
##
## Eigenvalues for unconstrained axes:
##    PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8    PC9
## 186.19  88.46  38.19  18.40  12.84  10.55   5.52   4.52   1.09
```

The `cca.object`

- Objects of class `"cca"` are complex with many components
- Entire class described in `?cca.object`
- Depending on what analysis performed some components may be `NULL`
- Used for (C)CA, PCA, RDA, CAP (`capscale()`), and dbRDA (`dbrda()`)

The `cca.object`

`cca1` has a large number of components

- `$call` how the function was called
- `$grand.total` in (C)CA sum of `rowsum`
- `$rowsum` the row sums
- `$colsum` the column sums
- `$tot.chi` total inertia, sum of Eigenvalues
- `$pCCA` Conditioned (partial-ed out) components
- `$CCA` Constrained components
- `$CA` Unconstrained components
- `$method` Ordination method used
- `$inertia` Description of what inertia is

The `cca.object`

Depending on how one called `cca()` etc some of these components will be `NULL`

`$pCCA` is only filled in if a *partial* constrained ordination fitted

`rda()` returns objects with classes `"rda"` and `"cca"`, but in most cases those objects work like those of class `"cca"`

The Eigenvalues and axis scores are now spread about the `$CA` and `$CCA` components (also `$pCCA` if a *partial* CCA)

Thankfully we can use *extractor* functions to get at such things

Eigenvalues

Use `eigenvals()` to extract Eigenvalues from a fitted ordination object

```
eigenvals(cca1)
```

```
##      CCA1      CCA2      CCA3      CCA4      CCA5      CCA6      CCA7      CCA8
## 0.4388704 0.2917753 0.1628465 0.1421302 0.1179519 0.0890291 0.0702945 0.0583592
##      CCA9     CCA10     CCA11     CCA12     CCA13     CCA14       CA1       CA2
## 0.0311408 0.0132944 0.0083644 0.0065385 0.0061563 0.0047332 0.1977645 0.1419256
##       CA3       CA4       CA5       CA6       CA7       CA8       CA9
## 0.1011741 0.0707868 0.0533034 0.0332994 0.0188676 0.0151044 0.0094876
```

Example

- Fit a CCA model to the lichen pasture data. The model should include, N, P, and K only.
- Save the model in object `mycca1`
- How much variance is explained by this model?
- Extract the eigenvalues, how many constrained axes are there?

```r
library("vegan")
data(varechem, varespec)
```

```r
mycca1 ← cca(varespec ~ N + P + K, data = varechem)
mycca1
```

```
## Call: cca(formula = varespec ~ N + P + K, data = varechem)
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Constrained    0.4464     0.2143    3
## Unconstrained  1.6368     0.7857   20
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##    CCA1    CCA2    CCA3
## 0.19309 0.16271 0.09060
##
## Eigenvalues for unconstrained axes:
##    CA1    CA2    CA3    CA4    CA5    CA6    CA7    CA8
## 0.4495 0.2870 0.1877 0.1675 0.1280 0.1050 0.0750 0.0629
## (Showing 8 of 20 unconstrained eigenvalues)
```

```r
ev ← eigenvals(mycca1, model = "constrained")
head(ev)
```

```
##       CCA1       CCA2       CCA3
## 0.19308594 0.16271109 0.09060228
```

```r
length(ev)
```

```
## [1] 3
```

Extracting axis scores

To extract a range of scores from a fitted ordination use `scores()`

- takes an ordination object as the first argument
- `choices` — which axes? Defaults to `c(1,2)`
- `display` — which type(s) of scores to return
  - `"sites"` or `"wa"`: scores for samples in response matrix
  - `"species"`: scores for variables/columns in response
  - `"lc"`: linear combination site scores
  - `"bp"`: biplot scores (coords of arrow tip)
  - `"cn"`: centroid scores (coords of factor centroids)

Extracting axis scores

```
str(scores(cca1, choices = 1:4, display = c("species","sites")), max = 1)
```

```
## List of 2
##  $ species: num [1:44, 1:4] 0.0753 -0.1813 -1.0535 -1.2774 -0.1526 ...
##   ..- attr(*, "dimnames")=List of 2
##  $ sites  : num [1:24, 1:4] 0.178 -0.97 -1.28 -1.501 -0.598 ...
##   ..- attr(*, "dimnames")=List of 2
```

```
head(scores(cca1, choices = 1:2, display = "sites"))
```

```
##          CCA1       CCA2
## 18  0.1784733 -1.0598842
## 15 -0.9702382 -0.1971387
## 24 -1.2798478  0.4764498
## 27 -1.5009195  0.6521559
## 23 -0.5980933 -0.1840362
## 19 -0.1102881  0.7143142
```

Scalings...

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 via the `scaling` argument

Scalings...

- `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
- `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

```
scores(cca1, choices = 1:2, display = "species", scaling = 3)
```

Scalings...

Thankfully we can use alternative descrpitors to extract scores:

- `"none"`
- `"sites"`
- `"species"`
- `"symmetric"`

Two modifiers select negative scores depending on whether the model is CCA or RDA:

- `hill = TRUE`
- `correlation = TRUE`

Example

- Using the CCA model you fitted, extract the site scores for axes 2 and 3 with Hill's scaling, focusing on the sites

```
scrs ← scores(mycca1, display = "sites", choices = c(2,3),
              scaling = "sites", hill = TRUE)
head(scrs)
```

```
##            CCA2        CCA3
## 18  0.21507383 -0.22617222
## 15 -0.53564592 -0.14736699
## 24 -0.28328352 -0.56306912
## 27 -0.79825273 -0.35205393
## 23 -0.06029273 -0.09438971
## 19  0.04742753  0.09586591
```

Partial constrained ordinations

Partial constrained ordinations remove the effect of one or more variables *then* fit model of interest

Argument `Z` is used for a data frame of variables to partial out

```
pcca ← cca(X = varespec,
           Y = varechem[, "Ca", drop = FALSE],
           Z = varechem[, "pH", drop = FALSE])
```

Or with the formula interface use the `Condition()` function

```
pcca ← cca(varespec ~ Ca + Condition(pH), data = varechem) ## easier!
```

Partial constrained ordinations

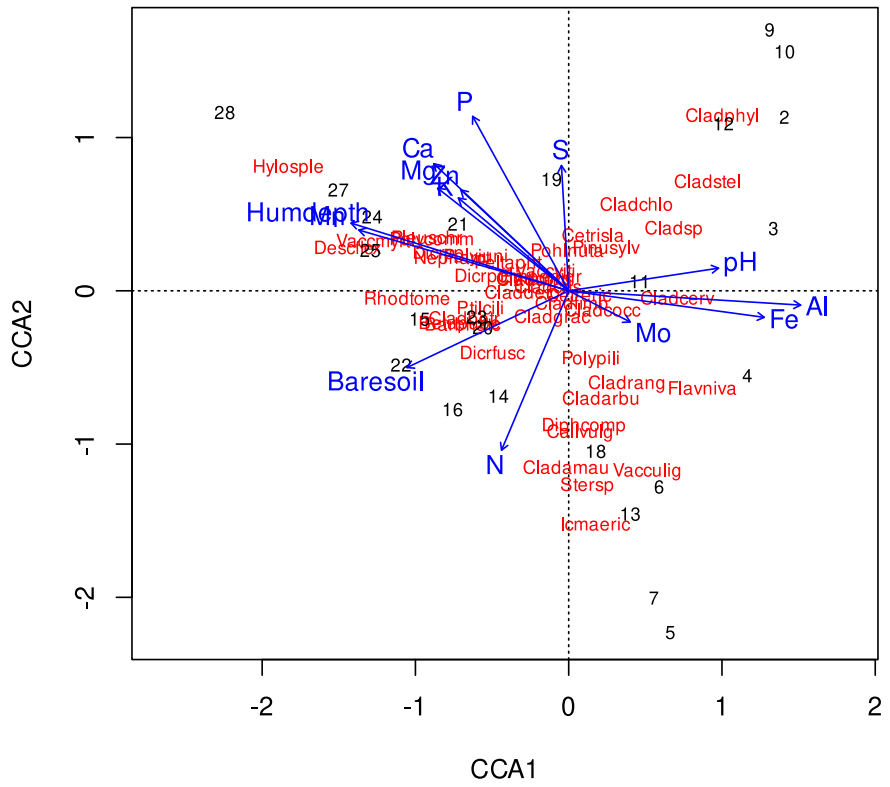
```
pcca ← cca(varespec ~ Ca + Condition(pH), data = varechem) ## easier!
pcca
```

```
## Call: cca(formula = varespec ~ Ca + Condition(pH), data = varechem)
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Conditional    0.1458     0.0700    1
## Constrained    0.1827     0.0877    1
## Unconstrained  1.7547     0.8423   21
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##    CCA1
## 0.18269
##
## Eigenvalues for unconstrained axes:
##    CA1    CA2    CA3    CA4    CA5    CA6    CA7    CA8
## 0.3834 0.2749 0.2123 0.1760 0.1701 0.1161 0.1089 0.0880
## (Showing 8 of 21 unconstrained eigenvalues)
```

Triplots

Triplots will generally produce a mess; we can really only display a couple of bits approximately anyway Trying to cram three things in is a recipe for a mess... but we can do it

```
plot(cca1)
```



LC vs WA scores

In constrained ordinations there are two sets of "site" scores

1. *linear combination* scores
2. *weighted average* (or weighted summation) scores

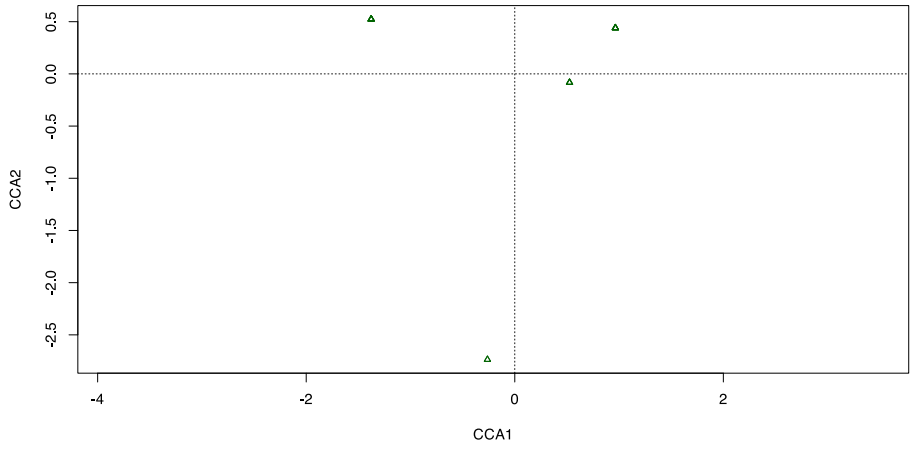
The LC scores are weighted & scaled versions of the constraints (predictors)

The WA scores are weighted averages of the species (responses) that are as close to the LC scores as possible

LC vs WA scores

Because the LC scores are based on the constraints you can get odd results if you plot them for some models

```
# example from Design Decision vignette
data(dune, dune.env, package = "vegan")
ord <- cca(dune ~ Moisture, data = dune.env)
plot(ord, display = "lc", type = "points")
```

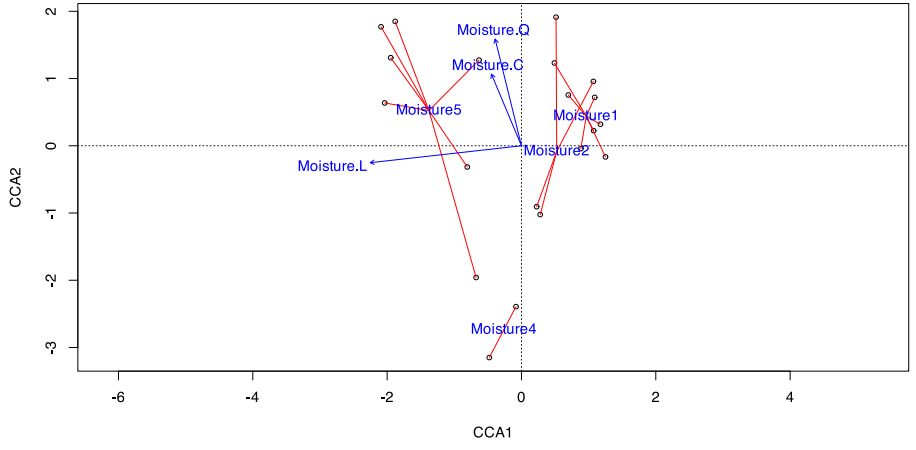


`Moisture` doesn't vary *within* observations of the same `Moisture` class (or level) so no variation in the LC scores for samples in each class.

LC vs WA scores

Weighted average scores include the variation in the species composition

```
# example from Design Decision vignette
data(dune, dune.env, package = "vegan")
ord ← cca(dune ~ Moisture, data = dune.env)
plot(ord, display = "wa", type = "points")
ordispider(ord, col = "red")
text(ord, display = "cn", col = "blue")
```



Model building

Building constrained ordination models

If we don't want to think it's easy to fit a poor model with many constraints — That's what I did with `cca1` and `rda1`

Remember, CCA and RDA are *just regression methods* — everything you know about regression applies here

A better approach is to *think* about the important variables and include only those

The formula interface allows you to create interaction or quadratic terms easily (though be careful with latter)

It also handles factor or class constraints automatically unlike the basic interface

Building constrained ordination models

```
vare.cca ← cca(varespec ~ Al + P*(K + Baresoil), data = varechem)
vare.cca
```

```
## Call: cca(formula = varespec ~ Al + P * (K + Baresoil), data =
## varechem)
##
##               Inertia Proportion Rank
## Total           2.083      1.000
## Constrained     1.046      0.502    6
## Unconstrained   1.038      0.498   17
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##   CCA1   CCA2   CCA3   CCA4   CCA5   CCA6
## 0.3756 0.2342 0.1407 0.1323 0.1068 0.0561
##
## Eigenvalues for unconstrained axes:
##     CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8
## 0.27577 0.15411 0.13536 0.11803 0.08887 0.05511 0.04919 0.03781
## (Showing 8 of 17 unconstrained eigenvalues)
```

Building constrained ordination models

For CCA, RDA etc we have little choice but to do

1. Fit well-chosen set of candidate models & compare, or
2. Fit a *full* model of well-chosen variables & then do stepwise selection

Automatic approaches to model building should be used cautiously!

The standard `step()` function can be used as *vegan* provides two helper methods, `deviance()` and `extractAIC()`, used by `step()`

vegan also provides methods for class `"cca"` for `add1()` and `drop1()`

Variance inflation factors

Linear dependencies between constraints can be investigated with VIF

VIF is a measure of how much the variance of $\hat{\beta}_j$ is inflated by presence of other covariates

Lots of rules of thumb

- VIF >= 20 indicates *strong collinearity* in constraints
- VIF >= 10 potentially of concern & should be looked at

Computed via `vif.cca()`

```
vif.cca(cca1)
```

```
##          N          P          K         Ca         Mg          S         Al         Fe
##   1.981742   6.028515  12.009357   9.925801   9.810609  18.378794  21.192739   9.127762
##         Mn         Zn         Mo   Baresoil   Humdepth         pH
##   5.380432   7.739664   4.320346   2.253683   6.012537   7.389267
```

Stepwise selection in CCA

`step()` uses AIC which is a fudge for RDA/CCA — use `ordistep()`

1. Define an upper and lower model scope, say the full model and the null model
2. To step from the lower scope or null model we use

```
upr ← cca(varespec ~ ., data = varechem)
lwr ← cca(varespec ~ 1, data = varechem)
set.seed(1)
mods ← ordistep(lwr, scope = formula(upr), trace = 0)
```

`trace = 0` is used here to turn off printing of progress

Permutation tests are used (more on these later); the theory for an AIC for ordination is somewhat loose

Stepwise selection in CCA

The object returned by `step()` is a standard `"cca"` object with an extra component `$anova`

```
mods
```

```
## Call: cca(formula = varespec ~ Al + P + K, data = varechem)
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Constrained    0.6441     0.3092    3
## Unconstrained  1.4391     0.6908   20
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##   CCA1   CCA2   CCA3
## 0.3616 0.1700 0.1126
##
## 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
## (Showing 8 of 20 unconstrained eigenvalues)
```

Stepwise selection in CCA

The $anova component contains a summary of the steps involved in automatic model building

```
mods$anova
```

```
##      Df    AIC      F Pr(>F)
## + Al  1 128.61 3.6749  0.005 **
## + P   1 127.91 2.5001  0.005 **
## + K   1 127.44 2.1688  0.050 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

1. Al added first, then
2. P, followed by
3. K, then stopped

Stepwise selection in CCA

Step-wise model selection is fairly fragile; if we start from the full model we won't end up with the same final model

```
mods2 ← step(upr, scope = list(lower = formula(lwr), upper = formula(upr)), trace = 0,
             test = "perm")
mods2
```

Stepwise selection in CCA

```
mods2 ← step(upr, scope = list(lower = formula(lwr), upper = formula(upr)), trace = 0,
              test = "perm")
mods2
```

```
## Call: cca(formula = varespec ~ P + K + Mg + S + Mn + Mo + Baresoil +
## Humdepth, data = varechem)
##
## -- Model Summary --
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Constrained    1.1165     0.5360    8
## Unconstrained  0.9667     0.4640   15
##
## Inertia is scaled Chi-square
##
## -- Eigenvalues --
##
## 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     CA9    CA10
```

Adjusted R^2 for ordination models

Ordinary R^2 is biased for the same reasons as for a linear regression

- adding a variable to constraints will increase R^2
- the larger the number of constraints in the model the larger R^2, is due to random correlations

Can attempt to account for this bias via an *adjusted* R^2 measure

Adjusted R^2 for ordination models

Can attempt to account for this bias via an *adjusted* R^2 measure

$$R^2_{adj} = 1 - \frac{n-1}{n-m-1}(1-R^2)$$

- n is number of samples m is number of constraints (model degrees of freedom)
- Can be used up to $\sim M > n/2$ before becomes too conservative
- Can be negative

```
RsquareAdj(cca1)
```

```
## $r.squared
## [1] 0.6919576
##
## $adj.r.squared
## [1] 0.2163163
```

Stepwise selection via adjusted R^2

Problems with stepwise selection are myriad. Affects RDA, CCA, etc

Blanchet *et al* (2008) proposed a two-step solution for models where R^2_{adj} makes sense

Stepwise selection via adjusted R^2

- *Global test* of all constraints
  - Proceed **only** if this test is significant
  - Helps prevent inflation of overall type I error
- Proceed with forward selection, but with *two* stopping rules
  - Usual significance threshold α
  - The global R^2_{adj}
  - Stop if next candidate model is non-significant or if R^2_{adj} exceeds the global R^2_{adj}

Available in `ordiR2step()`

Stepwise selection via adjusted R^2

```
ordiR2step(lwr, upr, trace = FALSE)
```

```
## Call: cca(formula = varespec ~ Al + P + K, data = varechem)
##
##               Inertia Proportion Rank
## Total          2.0832     1.0000
## Constrained    0.6441     0.3092    3
## Unconstrained  1.4391     0.6908   20
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##   CCA1   CCA2   CCA3
## 0.3616 0.1700 0.1126
##
## 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
## (Showing 8 of 20 unconstrained eigenvalues)
```

Permutation tests

Permutation tests in vegan

RDA has lots of theory behind it, CCA bit less. However, ecological/environmental data invariably violate what little theory we have

Instead we use permutation tests to assess the *importance* of fitted models — the data are shuffled in some way and the model refitted to derive a Null distribution under some hypothesis of *no effect*

Permutation tests in vegan

What *is* shuffled and *how* is of **paramount** importance for the test to be valid

- No conditioning (partial) variables then rows of the species data are permuted
- With conditioning variables, two options are available, both of which *permute residuals* from model fits
  - The *full model* uses residuals from model $Y = X + Z + \varepsilon$
  - The *reduced model* uses residuals from model $Y = Z + \varepsilon$
- In **vegan** which is used can be set via argument `model` with `"direct"`, `"full"`, and `"reduced"` respectively

Permutation tests in vegan

A test statistic is required, computed for observed model & each permuted model

vegan uses a pseudo F statistic

$$F = \frac{\chi^2_{model}/df_{model}}{\chi^2_{resid}/df_{resid}}$$

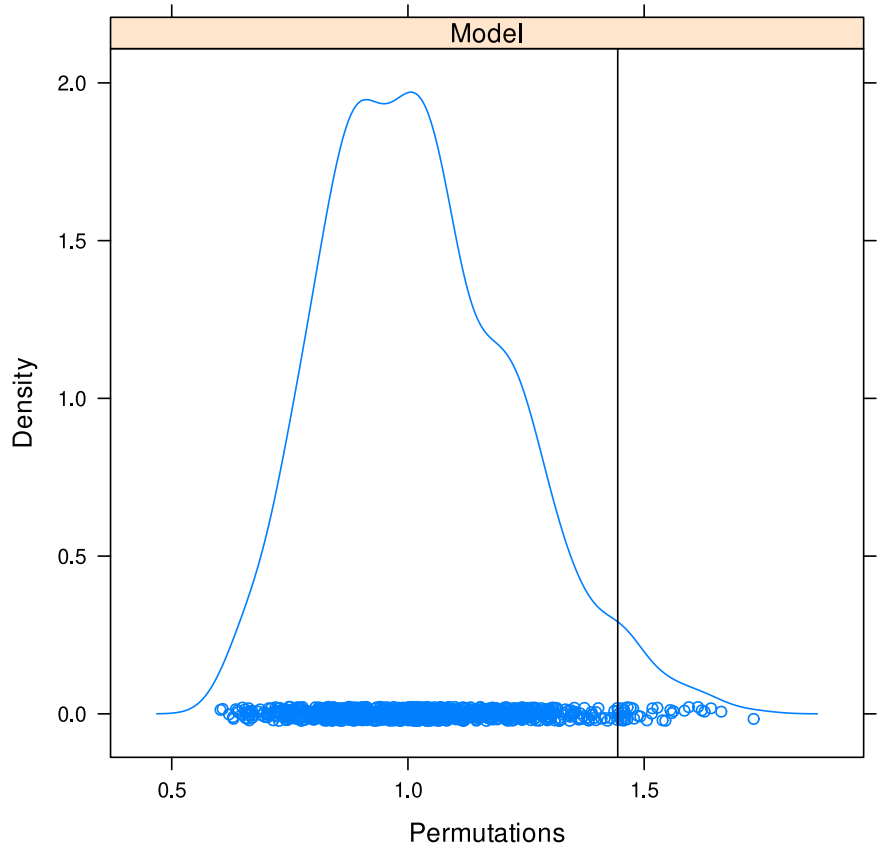
Evaluate whether F is unusually large relative to the null (permutation) distribution of F

Permutation tests in vegan

```
pstat ← permustats(anova(cca1))
summary(pstat)
```

```
##
##       statistic    SES   mean lower median  upper Pr(perm)
## Model    1.4441 2.0266 1.0382       1.0143 1.3989    0.035 *
## ---
## Signif. codes:  0 '**' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Interval (Upper - Lower) = 0.95)
```

```
densityplot(pstat)
```



Permutation tests in vegan: `anova()`

- The main user function is the `anova()` method
- It is an interface to the lower-level function `permutest.cca()`
- At its most simplest, the `anova()` method tests whether the **model** as a whole is significant

Permutation tests in vegan: anova( )

$$F = \frac{1.4415/14}{0.6417/9} = 1.4441$$

```
set.seed(42)
(perm ← anova(cca1))
```

```
## Permutation test for cca under reduced model
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = varespec ~ N + P + K + Ca + Mg + S + Al + Fe + Mn + Zn + Mo + Baresoil +
Humdepth + pH, data = varechem)
##          Df ChiSquare      F Pr(>F)
## Model    14   1.44148 1.4441  0.029 *
## Residual  9   0.64171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Permutation tests in vegan: `anova()`

`anova.cca()` has a number of arguments

```
args(anova.cca)
```

```
## function (object, ..., permutations = how(nperm = 999), by = NULL,
##     model = c("reduced", "direct", "full"), parallel = getOption("mc.cores"),
##     strata = NULL, cutoff = 1, scope = NULL)
## NULL
```

`object` is the fitted ordination

`permutations` controls what is permuted and how

`by` determines what is tested; the default is to test the model

Types of permutation test in vegan

A number of types of test can be envisaged

- Testing the overall significance of the model
- Testing constrained (canonical) axes
- Testing individual model terms *sequentially*
- The *marginal* effect of a single variable

The first is the default in `anova()`

The other three can be selected via the argument `by`

Testing canonical axes

- The constrained (canonical) axes can be individually tests by specifying `by = "axis"`
- The first axis is tested in terms of variance explained compared to residual variance
- The second axis is tested after partialling out the first axis...
- and so on

Testing canonical axes

```
set.seed(1)
anova(mods, by = "axis")
```

```
## Permutation test for cca under reduced model
## Forward tests for axes
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = varespec ~ Al + P + K, data = varechem)
##          Df ChiSquare      F Pr(>F)
## CCA1      1   0.36156 5.0249  0.001 ***
## CCA2      1   0.16996 2.3621  0.034 *
## CCA3      1   0.11262 1.5651  0.142
## Residual 20   1.43906
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Testing terms sequentially

- The individual terms in the model can be tested using `by = "terms"`
- The terms are assessed in the order they were specified in the model, sequentially from first to last
- Test is of the additional variance explained by adding the kth variable to the model
- **Ordering of the terms** will affect the results

Testing terms sequentially

```
set.seed(5)
anova(mods, by = "terms")
```

```
## Permutation test for cca under reduced model
## Terms added sequentially (first to last)
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = varespec ~ Al + P + K, data = varechem)
##          Df ChiSquare      F Pr(>F)
## Al        1   0.29817 4.1440  0.001 ***
## P         1   0.18991 2.6393  0.005 **
## K         1   0.15605 2.1688  0.018 *
## Residual 20   1.43906
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Testing terms marginal effects

- The marginal *effect* of a model term can be assessed using `by = "margin"`
- The marginal *effect* is the effect of a particular term when all other model terms are included in the model

Testing terms marginal effects

```
set.seed(10)
anova(mods, by = "margin")
```

```
## Permutation test for cca under reduced model
## Marginal effects of terms
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = varespec ~ Al + P + K, data = varechem)
##          Df ChiSquare      F Pr(>F)
## Al        1   0.31184 4.3340  0.001 ***
## P         1   0.16810 2.3362  0.016 *
## K         1   0.15605 2.1688  0.029 *
## Residual 20   1.43906
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Spring meadow vegetation

Example & data taken from Leps & Smilauer (2014), Case Study 2

Spring fen meadow vegetation in westernmost Carpathian mountains

```
# load vegan, dplyr & readr
library("vegan"); library("dplyr"); library("readr")

# load the data
spp ← read_csv("https://bit.ly/meadows-species") %>%
    rename("sample_id" = "...1") %>%
    tibble::column_to_rownames("sample_id")
env ← read_csv("https://bit.ly/meadows-env") %>%
    rename("sample_id" = "...1")
```

Spring meadow vegetation

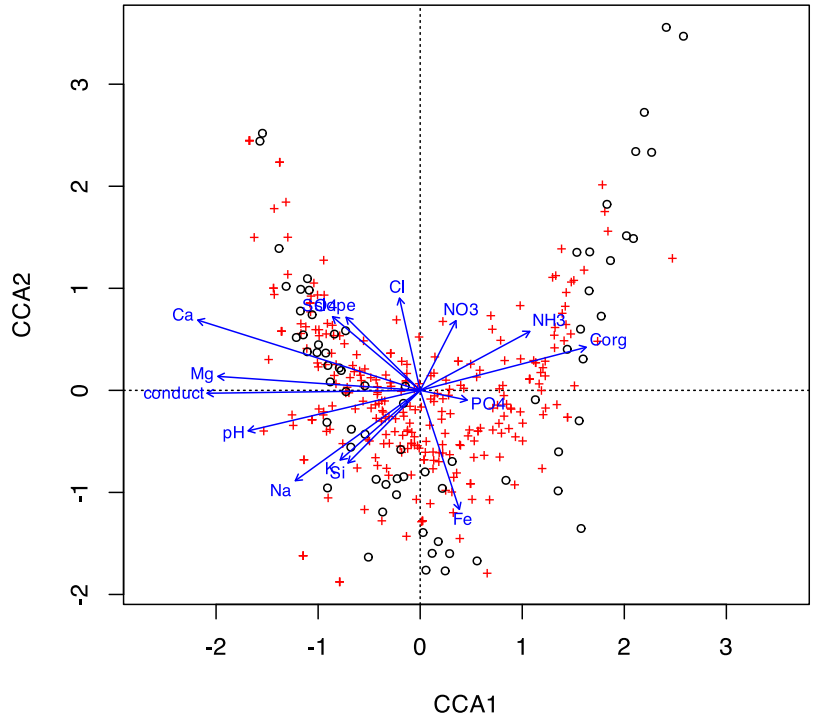
CCA a reasonable starting point as the gradient is long here (check with `decorana()` if you want)

```
m1 ← cca(spp ~ ., data = env)
set.seed(32)
anova(m1)
```

```
## Permutation test for cca under reduced model
## Permutation: free
## Number of permutations: 999
##
## Model: cca(formula = spp ~ Ca + Mg + Fe + K + Na + Si + SO4 + PO4 + NO3 + NH3 + Cl + Corg + pH +
conduct + slope, data = env)
##          Df ChiSquare     F Pr(>F)
## Model    15    1.5597 1.497  0.001 ***
## Residual 54    3.7509
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Spring meadow vegetation

```
plot(m1)
```



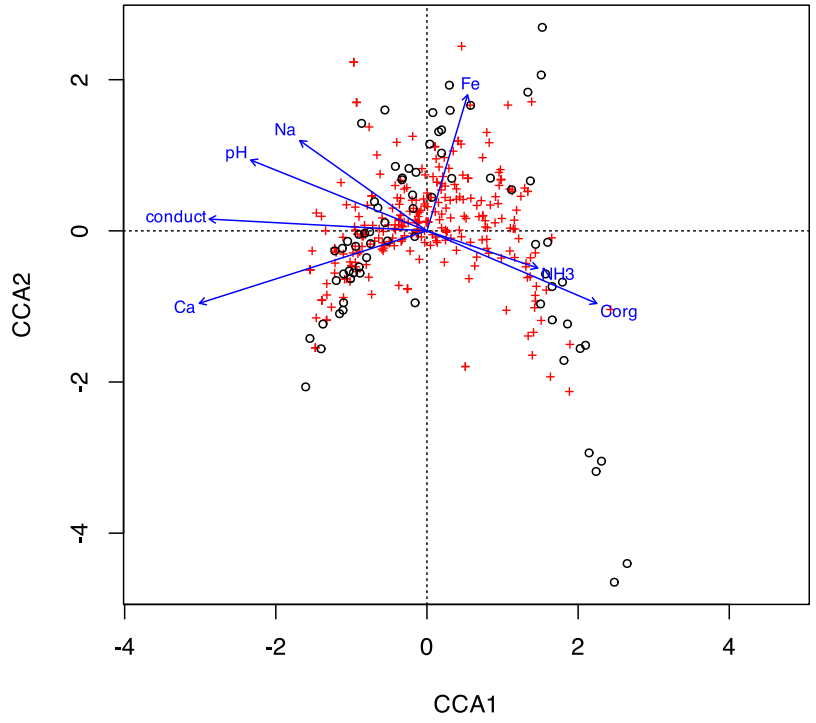
Spring meadow vegetation

```
set.seed(67)
lwr ← cca(spp ~ 1, data = env)
( m2 ← ordistep(lwr, scope = formula(m1), trace = FALSE) )
```

```
## Call: cca(formula = spp ~ Ca + conduct + Corg + Na + NH3 + Fe + pH,
## data = env)
##
##               Inertia Proportion Rank
## Total          5.3107     1.0000
## Constrained    0.9899     0.1864    7
## Unconstrained  4.3208     0.8136   62
## Inertia is scaled Chi-square
##
## Eigenvalues for constrained axes:
##   CCA1   CCA2   CCA3   CCA4   CCA5   CCA6   CCA7
## 0.4268 0.1447 0.1116 0.0936 0.0760 0.0719 0.0652
##
## Eigenvalues for unconstrained axes:
##     CA1     CA2     CA3     CA4     CA5     CA6     CA7     CA8
## 0.27251 0.19518 0.16703 0.14993 0.14606 0.14168 0.13292 0.12154
## (Showing 8 of 62 unconstrained eigenvalues)
```

Spring meadow vegetation

```
plot(m2)
```



Spring meadow vegetation

```
m2$anova
```

```
##           Df    AIC      F Pr(>F)
## + Ca       1 453.14 4.7893  0.005 **
## + conduct  1 453.29 1.7915  0.005 **
## + Corg     1 453.61 1.6011  0.005 **
## + Na       1 453.93 1.5827  0.010 **
## + NH3      1 454.36 1.4507  0.020 *
## + Fe       1 454.89 1.3386  0.040 *
## + pH       1 455.46 1.2756  0.040 *
## ---
## Signif. codes:  0 '**' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Spring meadow vegetation

Alternative is RDA with a transformation

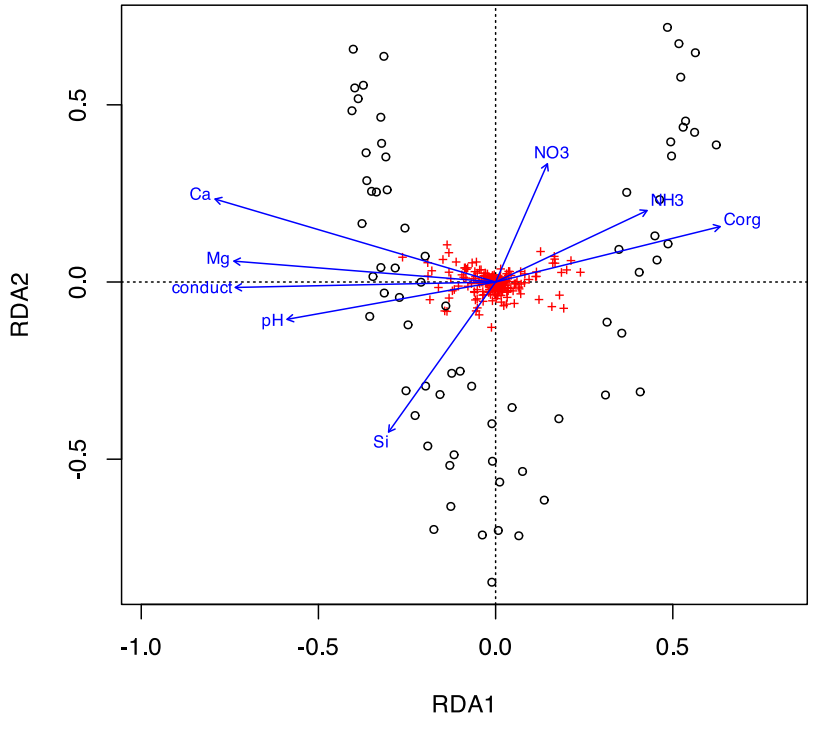
```
spph ← decostand(spp, method = "hellinger")
m3 ← rda(spph ~ ., data = env)
lwr ← rda(spph ~ 1, data = env)
m4 ← ordistep(lwr, scope = formula(m3),
                trace = FALSE)
```

```
m4
```

```
## Call: rda(formula = spph ~ Ca + NH3 + conduct + Si + Corg + NO3 + pH +
## Mg, data = env)
##
##               Inertia Proportion Rank
## Total          0.6123     1.0000
## Constrained    0.1823     0.2977    8
## Unconstrained  0.4300     0.7023   61
## Inertia is variance
##
## Eigenvalues for constrained axes:
##    RDA1    RDA2    RDA3    RDA4    RDA5    RDA6    RDA7    RDA8
## 0.10572 0.02148 0.01224 0.01148 0.00945 0.00891 0.00696 0.00609
##
## Eigenvalues for unconstrained axes:
##     PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8
## 0.04311 0.03026 0.02030 0.01767 0.01649 0.01519 0.01383 0.01346
## (Showing 8 of 61 unconstrained eigenvalues)
```

Spring meadow vegetation

```
plot(m4)
```



Spring meadow vegetation

Stepwise using R^2_{adj}

```
m5 ← ordiR2step(lwr, scope = formula(m3), trace = FALSE)
m5$anova
```

```
##                   R2.adj Df     AIC       F Pr(>F)
## + Ca             0.12588  1 -41.779 10.9370  0.002 **
## + NH3            0.14628  1 -42.468  2.6242  0.002 **
## + conduct        0.16322  1 -42.925  2.3570  0.002 **
## + Si             0.17711  1 -43.164  2.1136  0.002 **
## + Corg           0.18518  1 -42.940  1.6442  0.014 *
## + NO3            0.19257  1 -42.680  1.5853  0.010 **
## + pH             0.19966  1 -42.417  1.5583  0.012 *
## <All variables> 0.20332
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

PERMANOVA

MANOVA

MANOVA is the multivariate form of ANOVA

- Multivariate response data
- Categorical predictor variables

Decompose variation in the responses into

1. variation within groups
2. variation between groups

Test to see if two is unusually large relative to H_0

PERMANOVA

Doing that test requires lots of assumptions that rarely hold for ecological data

PERMANOVA: Permutational multivariate analysis of variance

Avoids most of these issues through the use of permutation tests

Directly decomposes a dissimilarity matrix into

1. variation within groups
2. variation between groups

PERMANOVA *sensu stricto*

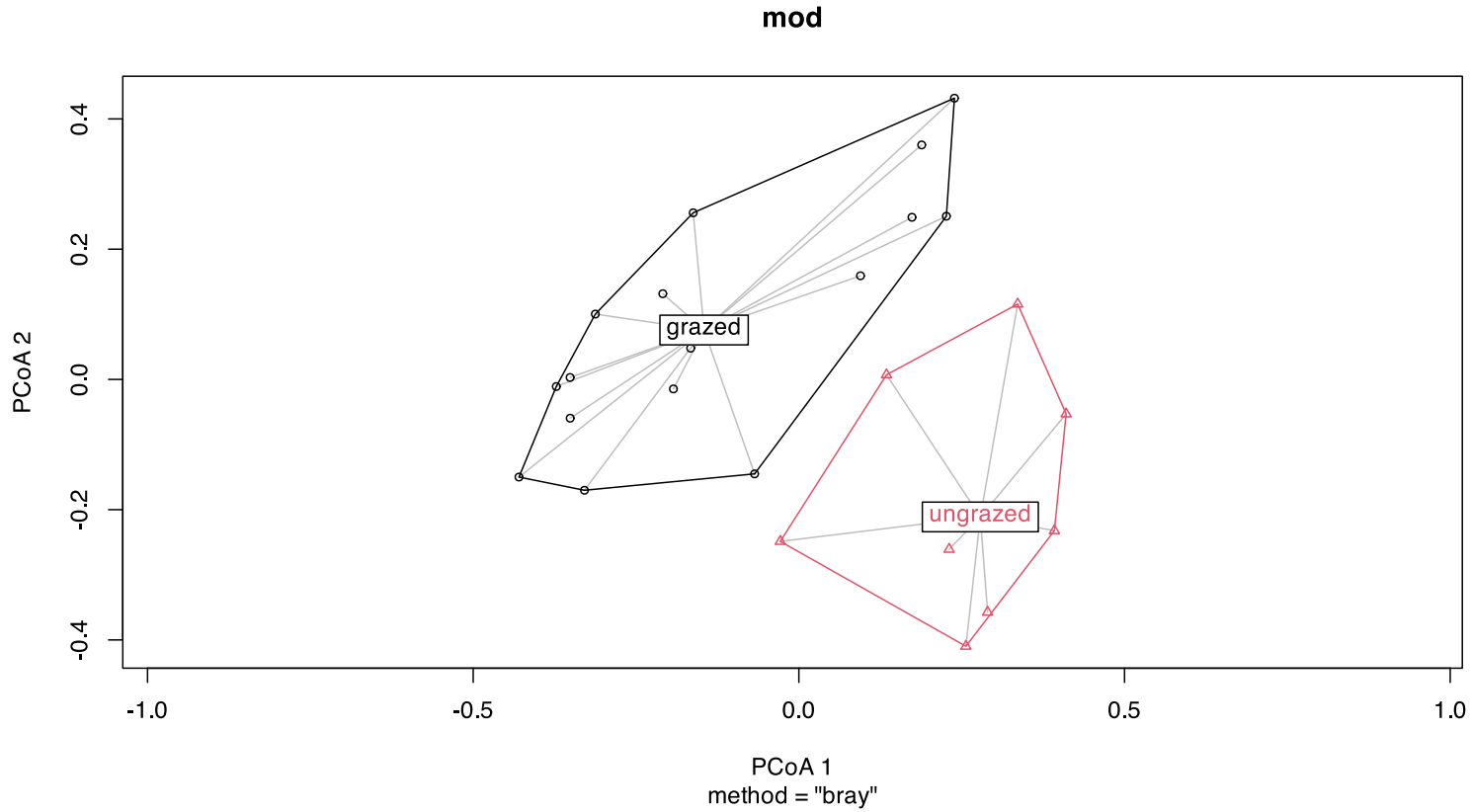
vegan has four different ways to do essentially do this kind of analysis

1. `adonis()` — implements Anderson (2001)
2. `adonis2()` — implements McArdle & Anderson (2001)
3. `dbrda()` — implementation based on McArdle & Anderson (2001)
4. `capscale()` — implements Legendre & Anderson (1999)

Be careful with `adonis()` as it allows only sequential tests

A difference between the functions is how they treat negative eigenvalues

The PERMANOVA idea



PERMANOA — adonis2()

```
data(dune, dune.env)
adonis2(dune ~ Management * A1, data = dune.env, by = "terms")
```

```
## Permutation test for adonis under reduced model
## Terms added sequentially (first to last)
## Permutation: free
## Number of permutations: 999
##
## adonis2(formula = dune ~ Management * A1, data = dune.env, by = "terms")
##              Df SumOfSqs      R2      F Pr(>F)
## Management    3   1.4686 0.34161 3.2629  0.003 **
## A1            1   0.4409 0.10256 2.9387  0.021 *
## Management:A1 3   0.5892 0.13705 1.3090  0.243
## Residual     12   1.8004 0.41878
## Total        19   4.2990 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

PERMANOA — adonis2()

```
data(dune, dune.env)
adonis2(dune ~ A1 * Management, data = dune.env, by = "terms")
```

```
## Permutation test for adonis under reduced model
## Terms added sequentially (first to last)
## Permutation: free
## Number of permutations: 999
##
## adonis2(formula = dune ~ A1 * Management, data = dune.env, by = "terms")
##               Df SumOfSqs      R2      F Pr(>F)
## A1             1   0.7230 0.16817 4.8187  0.001 ***
## Management     3   1.1865 0.27600 2.6362  0.008 **
## A1:Management  3   0.5892 0.13705 1.3090  0.200
## Residual      12   1.8004 0.41878
## Total         19   4.2990 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

PERMANOA — adonis2()

```
data(dune, dune.env)
adonis2(dune ~ Management * A1, data = dune.env, by = "margin")
```

```
## Permutation test for adonis under reduced model
## Marginal effects of terms
## Permutation: free
## Number of permutations: 999
##
## adonis2(formula = dune ~ Management * A1, data = dune.env, by = "margin")
##               Df SumOfSqs      R2     F Pr(>F)
## Management:A1  3   0.5892 0.13705 1.309  0.214
## Residual      12   1.8004 0.41878
## Total         19   4.2990 1.00000
```

The interaction is the only term that isn't *marginal* to other terms; not significant

PERMANOA — adonis2()

```
adonis2(dune ~ Management + A1, data = dune.env, by = "margin")
```

```
## Permutation test for adonis under reduced model
## Marginal effects of terms
## Permutation: free
## Number of permutations: 999
##
## adonis2(formula = dune ~ Management + A1, data = dune.env, by = "margin")
##            Df SumOfSqs      R2      F Pr(>F)
## Management  3   1.1865 0.27600 2.4828  0.003 **
## A1          1   0.4409 0.10256 2.7676  0.016 *
## Residual   15   2.3895 0.55583
## Total      19   4.2990 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The dispersion problem

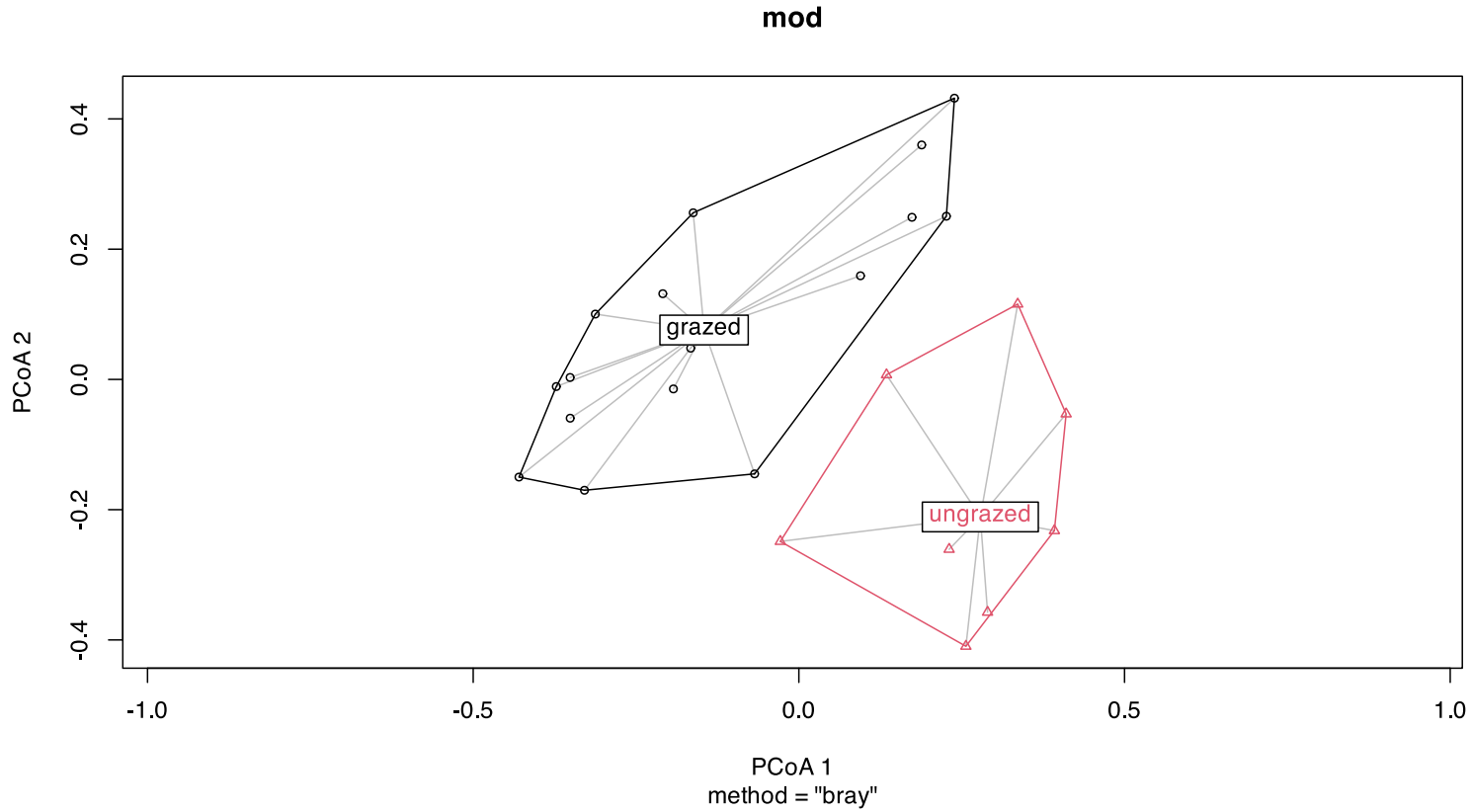
Anderson (2001) noted that PERMANOVA could confound *location* & *dispersion* effects

If one or more groups are more variable — dispersed around the centroid — than the others, this can result in a false detection of a difference of means — a *location* effect

Same problem affects *t* tests, ANOVA

Warton et al (2012) Anderson & Walsh (2013) Anderson *et al* (2017)

Dispersion


mod
grazed
ungrazed
PCoA 2
0.4
0.2
0.0
-0.2
-0.4
-1.0
-0.5
0.0
0.5
1.0
PCoA 1
method = "bray"

Test for dispersion effects

Marti Anderson (2006) developed a test for multivariate dispersions — PERMDISP2

1. Calculate how far each observation is from its group median (or centroid)
2. Take the absolute values of these distances-to-medians
3. Do an ANOVA on the absolute distances with the *groups* as covariates
4. Test the H_0 of equal absolute distances to median among groups using a permutation test

In *vegan* this is `betadisper()`

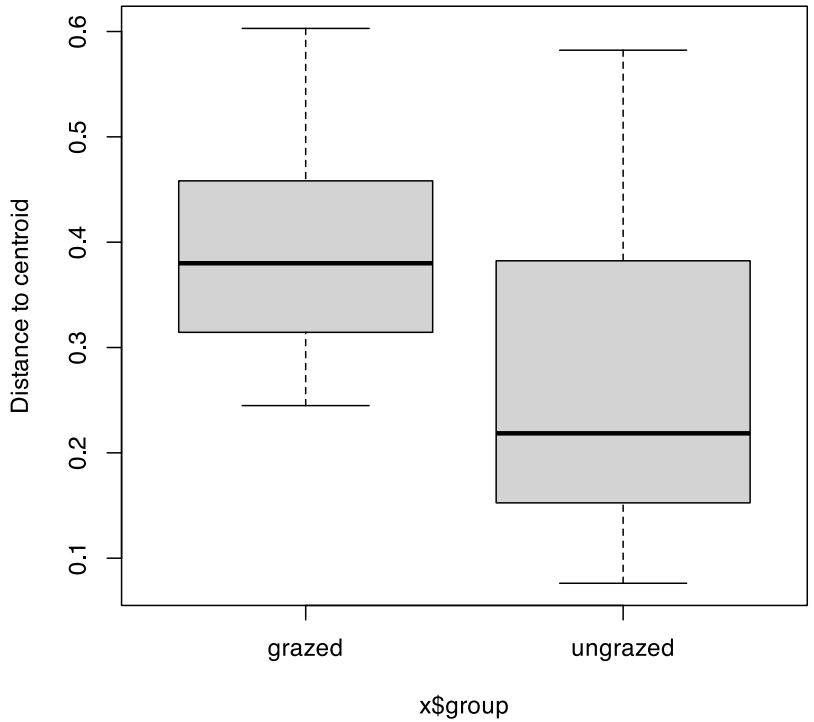
Test for dispersion effects

```
data(varespec)
dis ← vegdist(varespec) # Bray-Curtis distances
## First 16 sites grazed, remaining 8 sites ungrazed
groups ← factor(c(rep(1,16), rep(2,8)),
                labels = c("grazed","ungrazed"))

mod ← betadisper(dis, groups)
mod
```

```
##
##     Homogeneity of multivariate dispersions
##
## Call: betadisper(d = dis, group = groups)
##
## No. of Positive Eigenvalues: 15
## No. of Negative Eigenvalues: 8
##
## Average distance to median:
##   grazed ungrazed
##   0.3926   0.2706
##
## Eigenvalues for PCoA axes:
## (Showing 8 of 23 eigenvalues)
##  PCoA1  PCoA2  PCoA3  PCoA4  PCoA5  PCoA6  PCoA7
PCoA8
## 1.7552 1.1334 0.4429 0.3698 0.2454 0.1961 0.1751
0.1284
```

```
boxplot(mod)
```

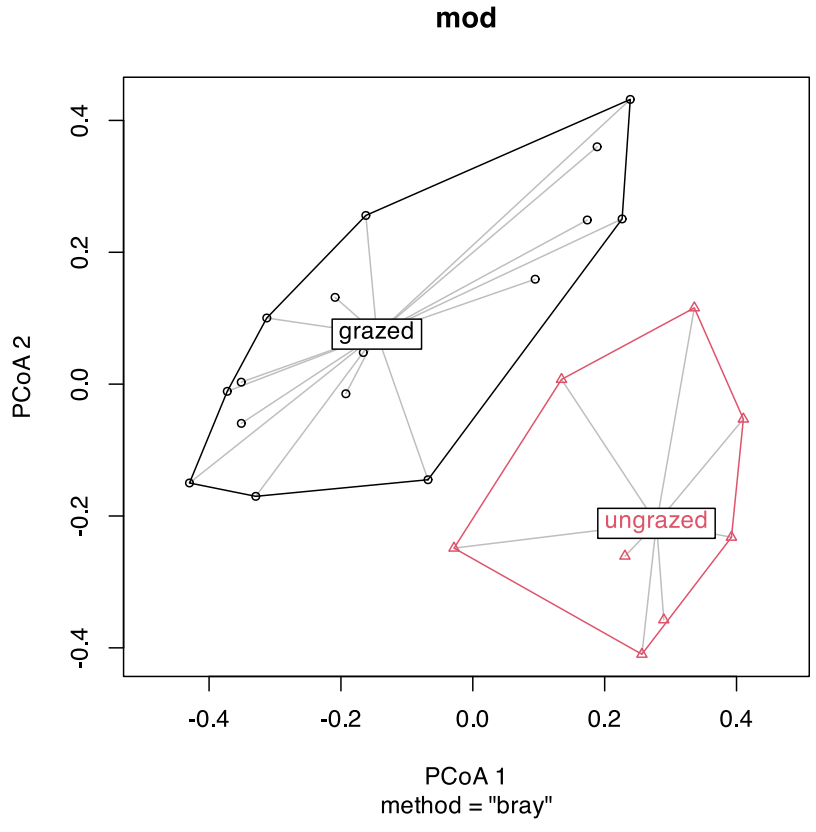


Test for dispersions

```
set.seed(25)
permutest(mod)
```

```
##
## Permutation test for homogeneity of multivariate
dispersions
## Permutation: free
## Number of permutations: 999
##
## Response: Distances
##           Df  Sum Sq  Mean Sq      F N.Perm Pr(>F)
## Groups     1 0.07931 0.079306 4.6156    999  0.045 *
## Residuals 22 0.37801 0.017182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
' ' 1
```

```
plot(mod)
```



Test for dispersions

```
set.seed(4)
permutest(mod, pairwise = TRUE)
```

```
##
## Permutation test for homogeneity of multivariate dispersions
## Permutation: free
## Number of permutations: 999
##
## Response: Distances
##           Df  Sum Sq  Mean Sq      F N.Perm Pr(>F)
## Groups     1 0.07931 0.079306 4.6156    999  0.036 *
## Residuals 22 0.37801 0.017182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Pairwise comparisons:
## (Observed p-value below diagonal, permuted p-value above diagonal)
##           grazed ungrazed
## grazed              0.043
## ungrazed 0.04295
```

Test for locations with non-equal dispersion?

Marti Anderson & colleagues (2017) have proposed a solution that is related to the Berens-Fisher problem

This is in Primer but not yet in *vegan*

https://github.com/vegandevs/vegan/issues/344

Distance-based RDA

Distance-based RDA

Multiple models that all do something similar

1. `adonis()` (deprecated)
2. `adonis2()`
3. `capscale()`
4. `dbrda()`

They all do essentially the same thing, but they do it differently & have slightly different behaviour

Distance-based RDA

Distance-based RDA (db-RDA) is a constrained form of principal coordinates analysis (PCO)

It is similar to RDA but allows for non-Euclidean dissimilarity indices

In *vegan*, db-RDA is implemented in `dbrda()`

Constrained analysis of principal coordinates

`capscale()` is *another* constrained form of PCO due to Legendre & Anderson (1999)

It is *very* similar to `dbrda()`

Constrained analysis of principal coordinates

`capscale()` works by

1. convert the response data into dissimilarities
2. apply PCO on the dissimilarities, take the PCO sample (site) scores as *new* response data
3. fit `rda()` to the *new* response data and predictor variables as constraints

Essentially, we embed the dissimilarities in a Euclidean space using PCO, and then we use RDA on this highly transformed response data

Distance-based RDA

db-RDA foregoes step 2., and directly decomposes the dissimilarities into components explained by each term in the model

Negative eigenvalues resulting from non-metric dissimilarity coefficients are handled via

1. square-root transform of the dissimilarities, or
2. adding a constant to the dissimilarities using methods `"lingoes"` (default, preferred) or `"cailliez"`

db-RDA is based on the ideas in McArdle & Anderson (2001)

Err... isn't that what `adonis2()` was developed to do?

Yes, but...

Distance-based RDA

`adonis2()` was a ground up redevelopment of the `adonis()` implementation and as such it retains many of the arguments and concepts of PERMANOVA, just updated to use the direct decomposition of dissimilarities

`dbrda()` inherits from `rda()` and `cca()` and as a result has expanded set of capability

`dbrda()` can use `Condition()` in the formula to fit partial db-RDA

`Condition()` is often needed to provide correct restricted permutation tests

Distance-based RDA

The equivalent model to `adonis2()` in `dbrda()`-form is

```
data(dune, dune.env)
dune_dbrda ← dbrda(dune ~ Management * A1, data = dune.env,
    distance = "bray")
```

because they have different default dissimilarities

References

- Anderson, M.J., 2001. A new method for non-parametric multivariate analysis of variance. Austral Ecol. 26, 32–46
- Anderson, M.J., 2006. Distance-based tests for homogeneity of multivariate dispersions. Biometrics 62, 245–253
- Anderson, M.J., Walsh, D.C.I., 2013. PERMANOVA, ANOSIM, and the Mantel test in the face of heterogeneous dispersions: What null hypothesis are you testing? Ecol. Monogr. 83, 557–574
- Anderson, M.J., Walsh, D.C.I., Robert Clarke, K., Gorley, R.N., Guerra-Castro, E., 2017. Some solutions to the multivariate Behrens-Fisher problem for dissimilarity-based analyses. Aust. N. Z. J. Stat. 59, 57–79
- Blanchet, F.G., Legendre, P., Borcard, D., 2008. Forward selection of explanatory variables. Ecology 89, 2623–2632
- Legendre, P., Anderson, M.J., 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. Ecol. Monogr. 69, 1–24
- McArdle, B.H., Anderson, M.J., 2001. Fitting Multivariate Models to Community Data: A Comment on Distance-Based Redundancy Analysis. Ecology 82, 290–297
- Warton, D.I., Wright, S.T., Wang, Y., 2012. Distance-based multivariate analyses confound location and dispersion effects. Methods Ecol. Evol. 3, 89–101