pcwOrd: (partial) (constrained) (weighted) ordination

Patrick Ewing

2020-03-04

Contents

1 Motivation

Many ordination packages exist for ecologists using R - `vegan` is excellent, for example. But none readily perform (to my knowledge) weighted, partial constrained ordinations.

The motivation for this is to analyze compositional ecological data - especially high throughput sequencing data - with methods that are robust, reproducible, and transparent. Compositional data contains only relative data (think relative abundance), and so requires a bit of extra care to analyze. The dominant approach is to perform a log-ratio transformation, then use principle components analysis - combined, this is called log-ratio analysis. For an enterance into this literature, see Greenacre and Aitchison (2002), Greenacre and Lewi (2009), and Quinn et al (2018).

In particular, Greenacre and Lewi (2009) suggest using weighted log-ratio analysis, as unweighted log-ratio analysis is susceptible to noise in low-abundance features. Raw values of low-abundance features have higher relative variance than raw values of high-abundance features; therefore, the log-ratios of low abundance features are also of lower certainty. When low-abundance features dominate a solution the distances and inferences made about samples is less robust to error and noise. Note that correspondence analysis also is suceptible to dominance by low-abundance features.

Weighted log-ratio analysis down-weights these uncertain, low-abundance features. As a result, the observed distances among samples should be more reproducible across experiments.

`vegan` can perform constrained and partialed log-ratio analysis if the community matrix is log-transformed beforehand. However, it does not provide a straightforward way to weight columns and working with ordination objects is not intuitive. `easyCODA` allows weighted and constrained log-ratio analysis, but does not allow partialing of nuisance effects and does not provide hypothesis testing. `pcwOrd` allows partialed, constrained, and weighted ordinations, and also provides utility functions for investigating ordination objects, visualization, and hypothesis testing. The code is written to be self-documenting. Maybe you'll agree.

This isn't an R package yet, so load the functions with `source()`:

```
libs = c('easyCODA',  # log-ratio analysis and CLR analysis
         'vegan')     # multivariate statistics
for(i in libs) {
  if (!require(i, character.only=TRUE)) {
    install.packages(i)
    if (!require(i, character.only=TRUE)) stop(paste("Cannot load", i))
  }}

# pcwOrd
load('../pcwOrd_0.1.Rdata')

# data
spider = readRDS('Spider.RDS')
```

This vignette will demonstrate the features of pcwOrd: ordination, hypothesis-testing, and visualization. I'll use the `spiders` dataset from `mvabund`. This dataset has two tables:

1. A `data.frame` of environmental variables, to which I'll add a categorical version of `soil.dry`*.
2. A `matrix` of community data.

Certain pcwOrd functions use the rownames of the environmental and community data to cross-reference, so we want to ensure both tables have rownames. Usually these are something meaningful, like sample IDs.

*technically this should be ordinal, but we'll treat it as categorical.

```
# environmental data
envi = as.data.frame(spider$x)

# categorical soil wetness
cats = c('dry', 'damp', 'wet', 'soaking')
cc = ceiling(envi$soil.dry)
envi$soil.cat = factor(cats[cc],
                       levels=cats)

# community data
comm = as.matrix(spider$abund)

# rownames
nobs = nrow(comm)
row_names = c(letters[], LETTERS[])[1:nobs]
rownames(envi) = row_names
rownames(comm) = row_names

str(envi)
```

```
## 'data.frame':    28 obs. of  7 variables:
##  $ soil.dry     : num  2.33 3.05 2.56 2.67 3.02 ...
##  $ bare.sand    : num  0 0 0 0 0 ...
##  $ fallen.leaves: num  0 1.79 0 0 0 ...
##  $ moss         : num  3.04 1.1 2.4 2.4 0 ...
##  $ herb.layer   : num  4.45 4.56 4.61 4.62 4.62 ...
##  $ reflection   : num  3.91 1.61 3.69 3 2.3 ...
##  $ soil.cat     : Factor w/ 4 levels "dry","damp","wet",..: 3 4 3 3 4 4 4 3 3 3 ...
```

```
comm[1:8, 1:8]
```

```
##   Alopacce Alopcune Alopfabr Arctlute Arctperi Auloalbi Pardlugu Pardmont
## a       25       10        0        0        0        4        0       60
## b        0        2        0        0        0       30        1        1
## c       15       20        2        2        0        9        1       29
## d        2        6        0        1        0       24        1        7
## e        1       20        0        2        0        9        1        2
## f        0        6        0        6        0        6        0       11
## g        2        7        0       12        0       16        1       30
## h        0       11        0        0        0        7       55        2
```

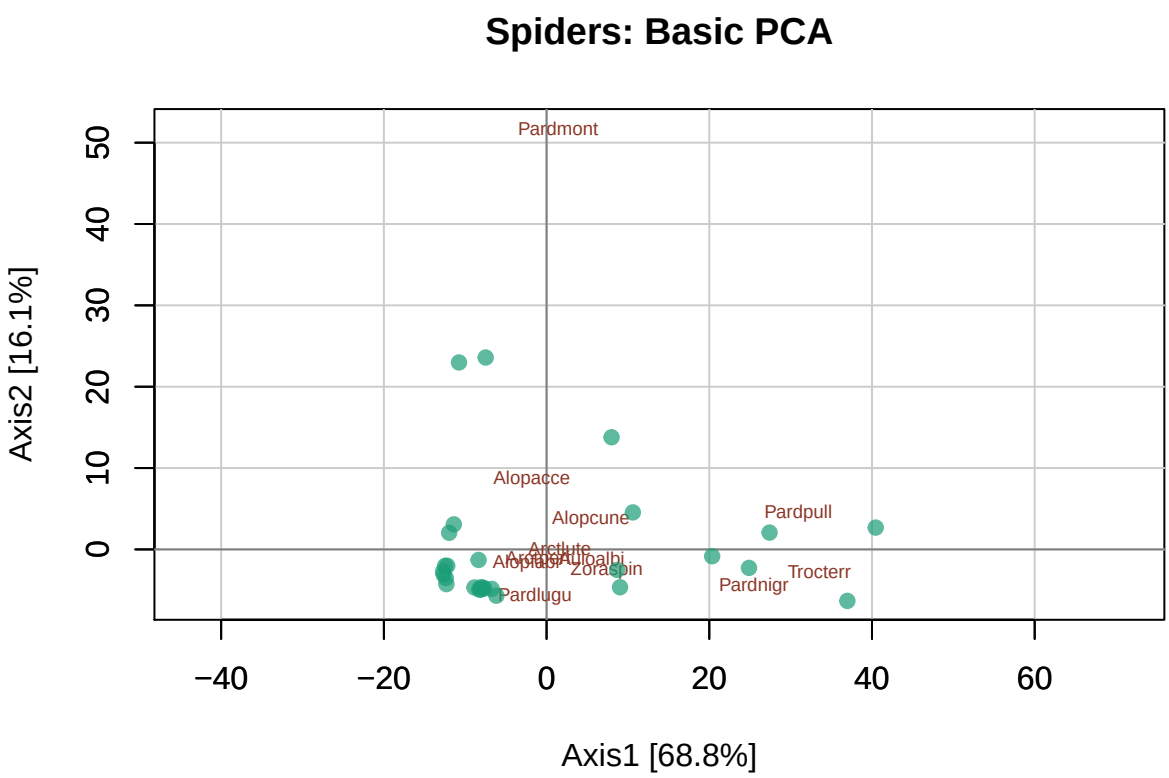
2 Basic Functions

2.1 Ordination

We'll first perform a basic principle components analysis of the community with `pcwOrd()` and visualize the results with `plot_ord`:

```
pca = pcwOrd(comm)
plot_ord(pca,
         main='Spiders: Basic PCA')
```

```
## Rescaling standard scores by 15.45
```



The `pca` object is a list of class `pcwOrd` that contains a number of items. See the documentation for `pcwOrd`. Important ones are: `'Y_scaled'` - the community matrix after centering, scaling, and/or weighting but before further analysis; `'unconstrained'` - the singular value decomosition matrices (left `'u'` and right `'v'`) and values (`'d'`). All relevant information about this ordination can be calculated from these values.

2.1.1 Eigenvalues

For example, if we want to calculate eigenvalues of this unconstrained ordination, we need to access the singular values (vector `d`) of the unconstrained solution:

```
pca$unconstrained$d^2
```

```
##   [1] 251.7432705  58.7674666  25.3873166  11.7224346   5.6704074   4.1539497
##   [7]   2.7649418   2.3272743   1.6733360   0.8060037   0.6634983   0.1884040
```

2.1.2 Axis Variances

We can view these same data by calling ord_variance, which summarizes the variance explained by each axis in the solution:

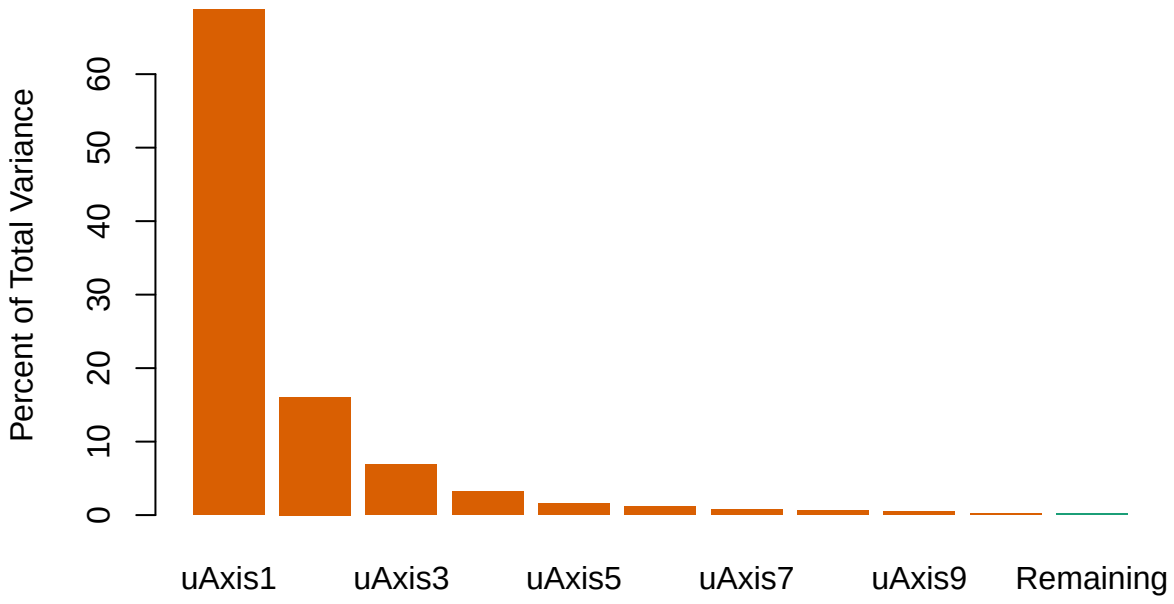
```
ord_variance(pca)
```

```
## $type
## [1] "variance"
##
## $summary
##              total partialed constrained unconstrained
## value     365.8683         0           0      365.8683
## pct_total 100.0000         0           0      100.0000
##
## $total
##              Total
## value     365.8683
## pct_group 100.0000
## pct_total 100.0000
##
## $partialed
##
## value
## pct_group
## pct_total
##
## $constrained
##
## value
## pct_group
## pct_total
##
## $unconstrained
##              uAxis1   uAxis2    uAxis3    uAxis4   uAxis5   uAxis6    uAxis7
## value     251.74327 58.76747 25.387317 11.722435 5.670407 4.153950 2.7649418
## pct_group  68.80707 16.06246  6.938922  3.204004 1.549849 1.135367 0.7557205
## pct_total  68.80707 16.06246  6.938922  3.204004 1.549849 1.135367 0.7557205
##              uAxis8    uAxis9   uAxis10   uAxis11    uAxis12
## value     2.3272743 1.6733360 0.8060037 0.6634983 0.18840395
## pct_group 0.6360962 0.4573602 0.2202989 0.1813489 0.05149502
## pct_total 0.6360962 0.4573602 0.2202989 0.1813489 0.05149502
```

2.1.3 Scree plots

We can also visualize variances across axes in a scree plot with `ord_scree()`:

```
ord_scree(pca)
```



2.1.4 Calculate scores

If you want to make your own biplots, you can calculate row, column, centeroid, and biplot scores by calling `ord_scores`. Scores can be principle, standard, or contribution - with the latter being raw singular values (either `'u'` or `'v'`). To make plotting easy, we can also add grouping information from the `envi` table:

```
grp = envi[, 'soil.cat', drop=FALSE] # preserve rownames
rowscores = ord_scores(pca,
                       choice='row',
                       scaling='principle',
                       axes=c(1:3),
                       add_grouping=grp)
head(rowscores)
```

```
##       Axis1     Axis2     Axis3 soil.cat
## a  7.988176 13.786265 -3.604174      wet
## b  9.030352 -4.652883 -5.718181  soaking
## c 10.606032  4.549642 -3.574303      wet
## d 24.881370 -2.272295 -8.576151      wet
## e 36.974719 -6.324696 19.094659  soaking
## f  8.683557 -2.525640 -1.763815  soaking
```

When plotting both standard and principle scores, often you'll need to rescale one of them for visualization:

```
colscores = ord_scores(pca,
                       choice='column',
                       scaling='standard',
                       axes=c(1:3))
scaled_colscores = scale_scores(colscores,
                                rowscores) # can also specify manual scaling factor
```

```
## Rescaling standard scores by 12.63
```

```
lapply(list(original = colscores,
            rescaled = scaled_colscores),
       head)
```

```
## $original
##                Axis1        Axis2       Axis3
## Alopacce -0.11660037  0.558436920  0.21907567
## Alopcune  0.35414730  0.243325952  0.17224233
## Alopfabr -0.15364830 -0.112002939  0.30171516
## Arctlute  0.10396386  0.008826392  0.04486536
## Arctperi -0.06877624 -0.081008292  0.12759310
## Auloalbi  0.35918227 -0.068503756 -0.42665830
##
## $rescaled
##               Axis1      Axis2     Axis3
## Alopacce -1.4732357  7.0558024  2.768002
## Alopcune  4.4746206  3.0744024  2.176267
## Alopfabr -1.9413330 -1.4151475  3.812145
## Arctlute  1.3135744  0.1115207  0.566870
## Arctperi -0.8689818 -1.0235328  1.612128
## Auloalbi  4.5382371 -0.8655391 -5.390791
```

2.1.5 Identify top features

Finally, it's useful to identify the top features. The default `scaling='contribution'` gives top contribution scores - i.e. the features that contribute most to the axes chosen. Here, we identify the spider taxa that have the five highest mean contributions across the first two axes:

```
top_scores(pca, n=5, choice='column', scaling='contribution', axes=c(1:2))
```

```
##          mean_contribution
## Pardmont         0.9681618
## Trocterr         0.6288081
## Pardpull         0.5841149
## Pardnigr         0.4835553
## Alopacce         0.1646834
## attr(,"axes")
## [1] 1 2
```

3 More ordinations

3.1 Log-ratio analysis

To perform log-ratio analysis:

1. Remove zeros from your data. For simplicity, I'll replace zeros with a pseudocount of 0.5. Note that I am *not* adding the pseudocount as `x + 0.5`, as this will distort the ratios between observations.
2. Close the community matrix, so that all abundances are relative abundances and `rowSums(x) = 1`.
3. Perform the log-ratio transformation of your choice. The centered log-ratio (CLR), where values are centered on the geometric mean, is a common choice.
4. Run principle components analysis on the log-ratio transformed matrix

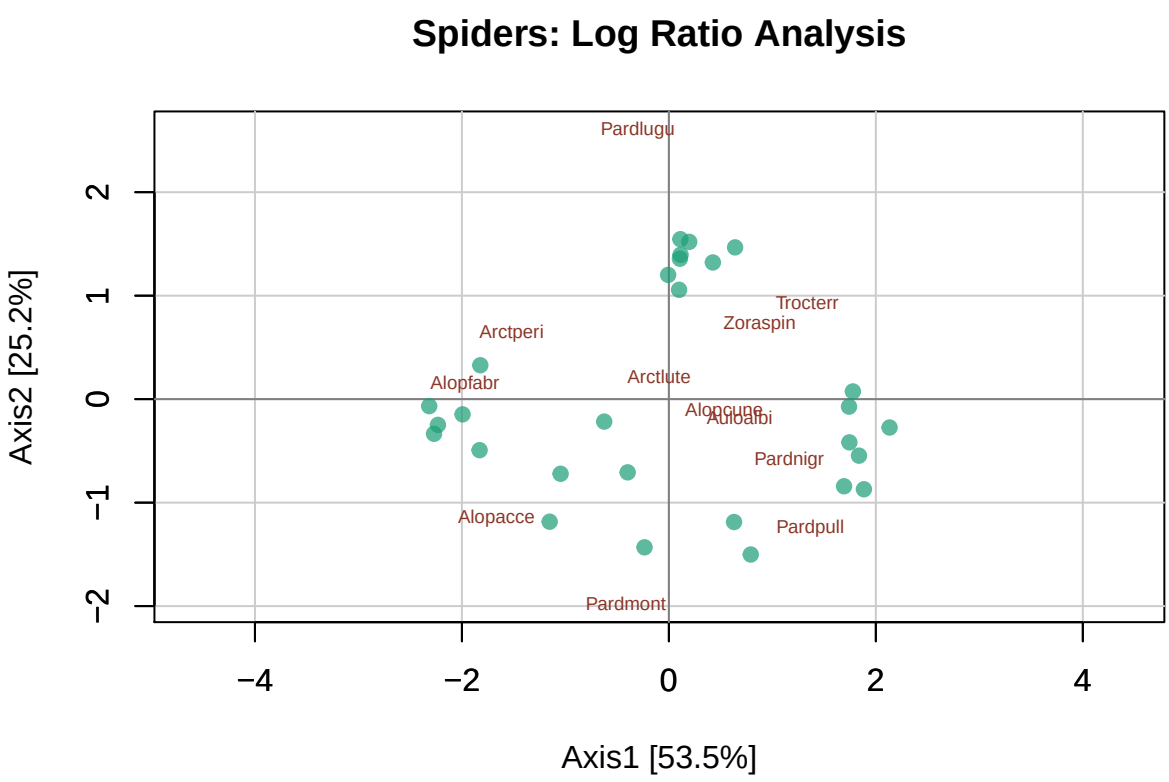
```
# Pseudocounts
cc = comm
cc[cc==0] = 0.5 # replace zeros without adding to all values!

# Close the community
closed_comm = sweep(cc, 1, rowSums(comm), '/') # close to relative abundance

# perform centered log-ratio
geo_mean = apply(closed_comm, 1,
                 function(x) {
                   exp(sum(log(x))/length(x))
                 })
lr_comm =  sweep(closed_comm, 1, geo_mean, '/')
lr_comm = log(lr_comm, 2)

lra = pcwOrd(lr_comm)
plot_ord(lra,
         main='Spiders: Log Ratio Analysis')
```

```
## Rescaling standard scores by 1.146
```



3.2 Weighted ordinations

To weight either rows or columns, for example by sequencing depth or column prevalence, you have three options:

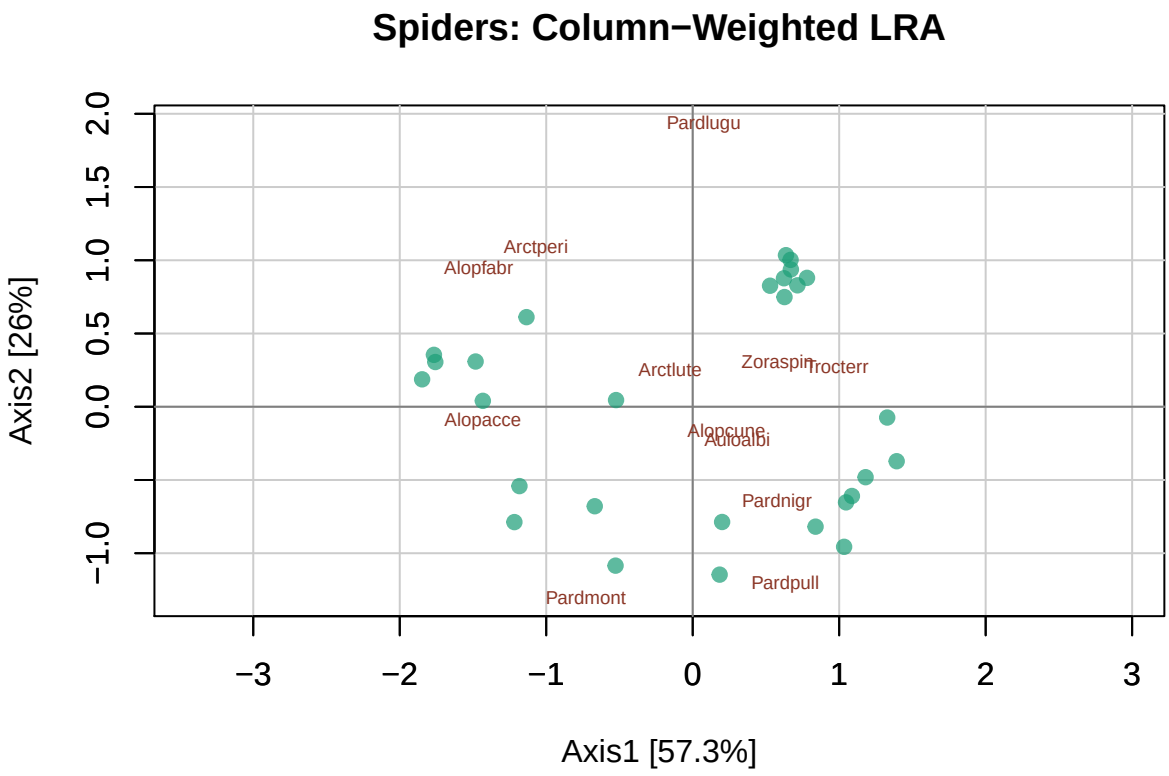
1. Tell pcwOrd to calculate weights automatically by setting `weight_rows=TRUE` or `weight_columns=TRUE`. This uses row or column masses as weights (via `rowSums`, for example).
2. Specify a vector of length `nrow()` or `ncol()` with the weights.
3. Input the result of `easyCODA::CLR()`, which returns a weighted CLR transformation and row-weights. This approach is demonstrated below:

Here is a weighted LRA:

```
wlr_comm = CLR(closed_comm) # CLR based on column-weighted geometric mean, plus column weights

w_lra = pcwOrd(wlr_comm)
plot_ord(w_lra,
         main='Spiders: Column-Weighted LRA')
```

```
## Rescaling standard scores by 0.9226
```



In this situation, the results of weighted and unweighted ordinations are approximately the same. See Greenacre and Lewi (2009) for a discussion of when weighted log-ratio analysis might be advantageous over a non-weighted log ratio analysis.

3.3 Constrained ordination

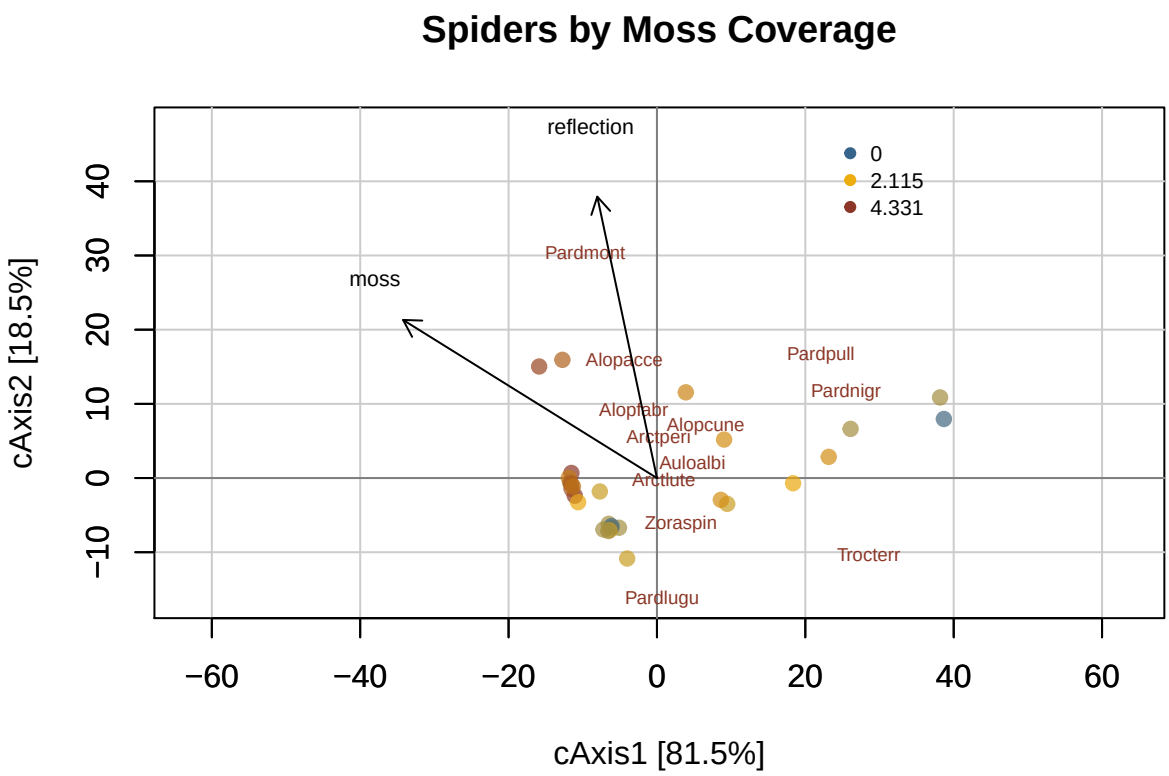
We might be interested in how these spiders associate with environmental variables. For example, we might be interested in whether moss and reflection structures spider communities. A constrained ordination like redundancy analysis will do this: pcwOrd will regress the community matrix against moss and reflectivity, and then ordinate the fitted values.

```
X = envi[, c('moss', 'reflection')] # preserve rownames
rda = pcwOrd(comm, X=X)

plot_ord(rda,
         main='Spiders by Moss Coverage',
         color_legend_position=c(0.8, 0))
```

```
## Rescaling standard scores by 13.4

## Rescaling standard scores by 48.24
```



In this plot, colors show moss coverage, the first column in X.

3.3.1 Significance testing

To test whether spiders vary with wetness class, run a PERMANOVA on the ordination with `permute_ord()`:

```
pp = permute_ord(rda)
pp
```

```
##   permute_on total_variance variance_after_partialing   fitted residuals num_df
## 1    partial       365.8683                 365.8683 98.72652  267.1418      2
##   denom_df   F_stat p_val F_perm
## 1       25 4.619575 0.004    999
```

`permute_ord()` has a number of permutation models that parallel the options in `vegan::permutest()` - and `permute_ord()` will return the same results as `vegan::permutest()` with the same randomization seed. We see a p-value of 0.004 after 999 permutations.

3.4 Partial ordination

Say we want to look at how moass affects spider communities independently of moisture class. We can partial this out as Z:

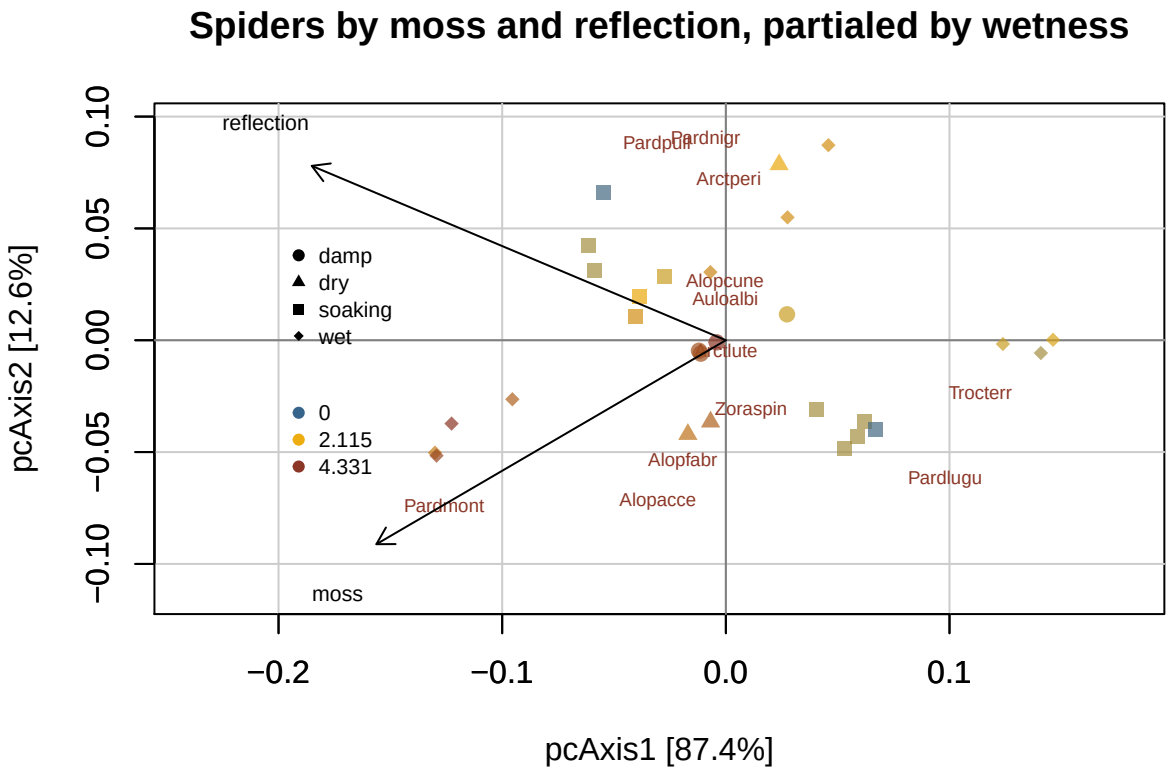
```
Z = envi[, 'soil.cat', drop=FALSE] # preserve rownames
X = envi[, c('moss', 'reflection'), drop=FALSE]

pcrda = pcwOrd(closed_comm, X, Z)

plot_ord(pcrda,
         main='Spiders by moss and reflection, partialed by wetness',
         shape_legend_position=c(-0.2, 0.05))
```

```
## Rescaling standard scores by 0.05837
```

```
## Rescaling standard scores by 0.2961
```



Each of the wetness categories is centered at zero, as expected due to partialing. Moss increases to the bottom left (darker red, and arrows). This model is highly significant:

```
permute_ord(pcrda)
```

```
##   permute_on total_variance variance_after_partialing      fitted   residuals
## 1    partial     0.01927485                0.01070139 0.004584283 0.006117103
##   num_df denom_df   F_stat p_val F_perm
## 1      2       22 8.243627 0.001    999
```

3.5 Partialed, constrained, weighted ordination

The reason for this package is to perform partial, constrained, weighted ordinations. Here is a weighted log-ratio analysis of the spider community, constrained by moss and reflection, with moisture category partialed out:

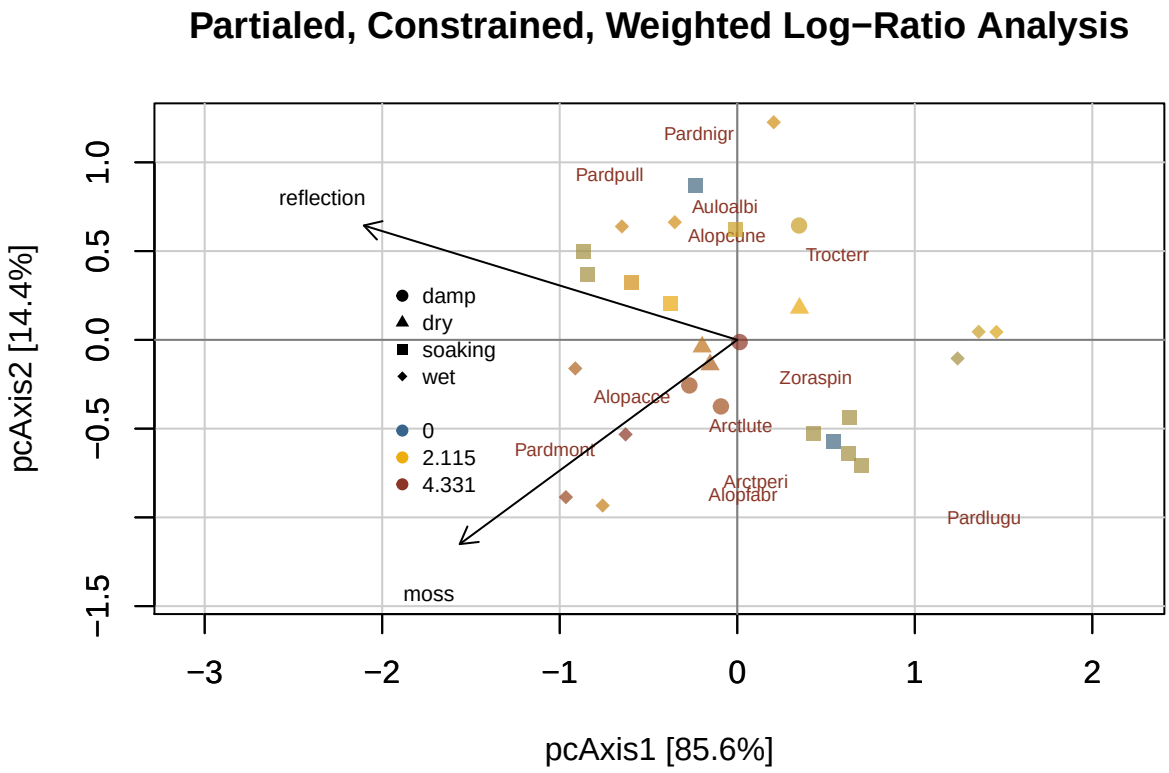
```
Y = CLR(closed_comm)

pcw_ord = pcwOrd(Y, X, Z)

plot_ord(pcw_ord,
         main='Partialed, Constrained, Weighted Log-Ratio Analysis',
         shape_legend_position=c(-2, 0.4))
```

```
## Rescaling standard scores by 0.7005

## Rescaling standard scores by 3.215
```



4 Comparison to `vegan` and `easyCODA`

`pcwOrd` gives the same results as comparable `vegan` and `easycCODA` functions.

4.1 Eigenvalues

Returning to eigenvalues: these eigenvalues are exactly the same as calculated by easyCODA's PCA, and solutions are the same:

```
easy_pca = PCA(comm, weight=FALSE)

rbind(pcwOrd = pca$unconstrained$d^2,
      easyCODA = easy_pca$sv^2
)
```

```
##              [,1]     [,2]     [,3]     [,4]     [,5]    [,6]     [,7]     [,8]
## pcwOrd   251.7433 58.76747 25.38732 11.72243 5.670407 4.15395 2.764942 2.327274
## easyCODA 251.7433 58.76747 25.38732 11.72243 5.670407 4.15395 2.764942 2.327274
##              [,9]     [,10]     [,11]    [,12]
## pcwOrd   1.673336 0.8060037 0.6634983 0.188404
## easyCODA 1.673336 0.8060037 0.6634983 0.188404
```

Eigenvalues are not identical to `vegan` eigenvalues, because `pcwOrd` (and `easyCODA`) weights the initial community matrix rows by `1/sqrt(nrow)`, while `vegan` weights rows by `1/sqrt(nrow-1)`. You can access `vegan`-style weighting, which will agree with vegan results:

```
veganlike_pca = pcwOrd(comm, as_vegan=TRUE)
vegan_pca = rda(comm)

rbind(pcwOrd = veganlike_pca$unconstrained$d^2,
      vegan = vegan_pca$CA$eig)
```

```
##             PC1      PC2      PC3      PC4      PC5     PC6      PC7      PC8
## pcwOrd 3132.805 731.3285 315.9311 145.8792 70.56507 51.6936 34.40816 28.96164
## vegan  3132.805 731.3285 315.9311 145.8792 70.56507 51.6936 34.40816 28.96164
##             PC9     PC10     PC11     PC12
## pcwOrd 20.82374 10.03027 8.256867 2.344583
## vegan  20.82374 10.03027 8.256867 2.344583
```

These weightings don't change relative variance or relative distances, only absolute variances and distances. Generally, we're concerned with relative variances and distances, so the choice of (equal) weighting doesn't matter.

```
list(
  coda_weightings = ord_variance(pca)$unconstrained,
  vegan_weightings = ord_variance(veganlike_pca)$unconstrained
)
```

```
## $coda_weightings
##              uAxis1   uAxis2    uAxis3    uAxis4   uAxis5   uAxis6    uAxis7
## value     251.74327 58.76747 25.387317 11.722435 5.670407 4.153950 2.7649418
## pct_group  68.80707 16.06246  6.938922  3.204004 1.549849 1.135367 0.7557205
## pct_total  68.80707 16.06246  6.938922  3.204004 1.549849 1.135367 0.7557205
##              uAxis8    uAxis9   uAxis10   uAxis11    uAxis12
## value     2.3272743 1.6733360 0.8060037 0.6634983 0.18840395
## pct_group 0.6360962 0.4573602 0.2202989 0.1813489 0.05149502
```

```
## pct_total 0.6360962 0.4573602 0.2202989 0.1813489 0.05149502
##
## $vegan_weightings
##               uAxis1    uAxis2     uAxis3     uAxis4    uAxis5    uAxis6
## value     3132.80514 731.32847 315.931051 145.879186 70.565070 51.693597
## pct_group   68.80707  16.06246   6.938922   3.204004  1.549849  1.135367
## pct_total   68.80707  16.06246   6.938922   3.204004  1.549849  1.135367
##               uAxis7     uAxis8     uAxis9    uAxis10   uAxis11    uAxis12
## value     34.4081650 28.9616352 20.8237366 10.0302687 8.2568673 2.34458251
## pct_group  0.7557205  0.6360962  0.4573602  0.2202989 0.1813489 0.05149502
## pct_total  0.7557205  0.6360962  0.4573602  0.2202989 0.1813489 0.05149502
```

4.2 Scores

```
ord_scores(pca, 'row', 'principle')
```

```
##        Axis1      Axis2
## a   7.988176 13.7862654
## b   9.030352 -4.6528827
## c  10.606032  4.5496420
## d  24.881370 -2.2722951
## e  36.974719 -6.3246958
## f   8.683557 -2.5256396
## g  40.458746  2.6835088
## h  -6.167524 -5.6887597
## i -11.395552  3.0854298
## j -11.973728  2.0464630
## k -10.765634 22.9846309
## l  -7.483546 23.5898261
## m  27.409257  2.0678738
## n  20.354007 -0.8378813
## o  -7.687032 -4.8091247
## p  -6.740881 -4.8547748
## q  -8.058080 -4.6416878
## r  -7.769638 -4.7163087
## s  -8.108993 -4.9559103
## t  -8.264546 -4.9285167
## u  -8.866147 -4.6802406
## v -12.383293 -3.5219133
## w -12.469082 -2.0121482
## x -12.652253 -3.0858499
## y  -8.360181 -1.2930062
## z -12.312196 -4.2679844
## A -12.721757 -2.7161932
## B -12.206151 -2.0078266
```

```
easy_pca$rowcoord[, 1:2]
```

```
##             [,1]       [,2]
##  [1,]  0.5034643  1.7983663
##  [2,]  0.5691487 -0.6069510
```

```
##  [3,]  0.6684578  0.5934836
##  [4,]  1.5681780 -0.2964123
##  [5,]  2.3303757 -0.8250327
##  [6,]  0.5472915 -0.3294601
##  [7,]  2.5499606  0.3500536
##  [8,] -0.3887156 -0.7420772
##  [9,] -0.7182182  0.4024827
## [10,] -0.7546585  0.2669534
## [11,] -0.6785169  2.9982583
## [12,] -0.4716594  3.0772038
## [13,]  1.7275011  0.2697463
## [14,]  1.2828355 -0.1092984
## [15,] -0.4844843 -0.6273322
## [16,] -0.4248521 -0.6332871
## [17,] -0.5078701 -0.6054906
## [18,] -0.4896907 -0.6152247
## [19,] -0.5110789 -0.6464798
## [20,] -0.5208828 -0.6429064
## [21,] -0.5587995 -0.6105197
## [22,] -0.7804718 -0.4594203
## [23,] -0.7858788 -0.2624771
## [24,] -0.7974233 -0.4025375
## [25,] -0.5269104 -0.1686678
## [26,] -0.7759908 -0.5567424
## [27,] -0.8018039 -0.3543171
## [28,] -0.7693072 -0.2619134
```

4.3 Permutation Tests

`vegan::permutest()` gives the same as `pcwOrd::permute_ord()`, if you set the same seed.

```
vegan_rda = rda(comm, X)
veganlike_rda = pcwOrd(comm, X, as_vegan=TRUE)

set.seed(445)
vegan_permutes = permutest(vegan_rda,
                           permutations=999)
set.seed(445)
pcw_permutes = permute_ord(veganlike_rda)
keep = c('fitted', 'residuals', 'num_df', 'denom_df', 'F_stat', 'p_val')

list(vegan = vegan_permutes,
     pcwOrd = pcw_permutes[keep])
```

```
## $vegan
##
## Permutation test for rda under reduced model
##
## Permutation: free
## Number of permutations: 999
##
## Model: rda(X = comm, Y = X)
## Permutation test for all constrained eigenvalues
```

```
##          Df Inertia      F Pr(>F)
## Model     2  1228.6 4.6196  0.004 **
## Residual 25  3324.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $pcwOrd
##     fitted residuals num_df denom_df   F_stat p_val
## 1 1228.597  3324.431      2       25 4.619575 0.004
```

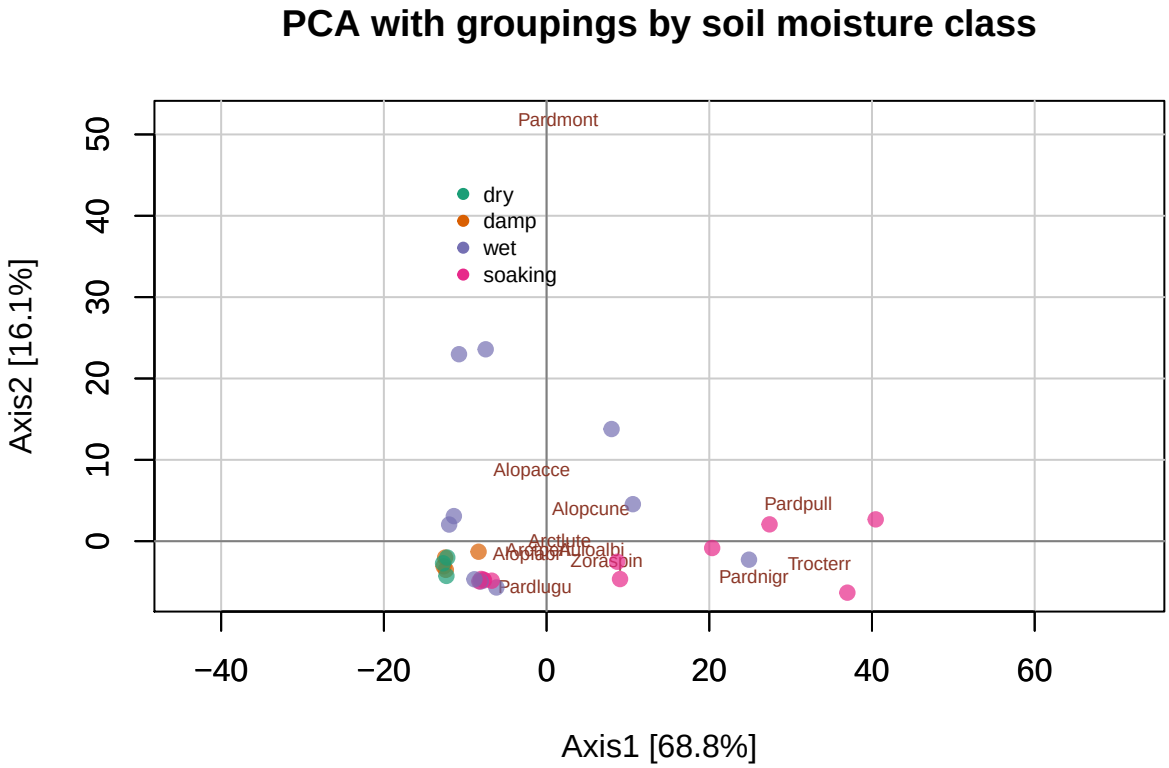
5 Additional plotting options

5.1 Added categories

Add groupings to any ordination. They can be continuous.

```
row_group = envi[, 'soil.cat', drop=FALSE]
plot_ord(pca,
         row_group=row_group,
         main='PCA with groupings by soil moisture class')
```

```
## Rescaling standard scores by 15.45
```

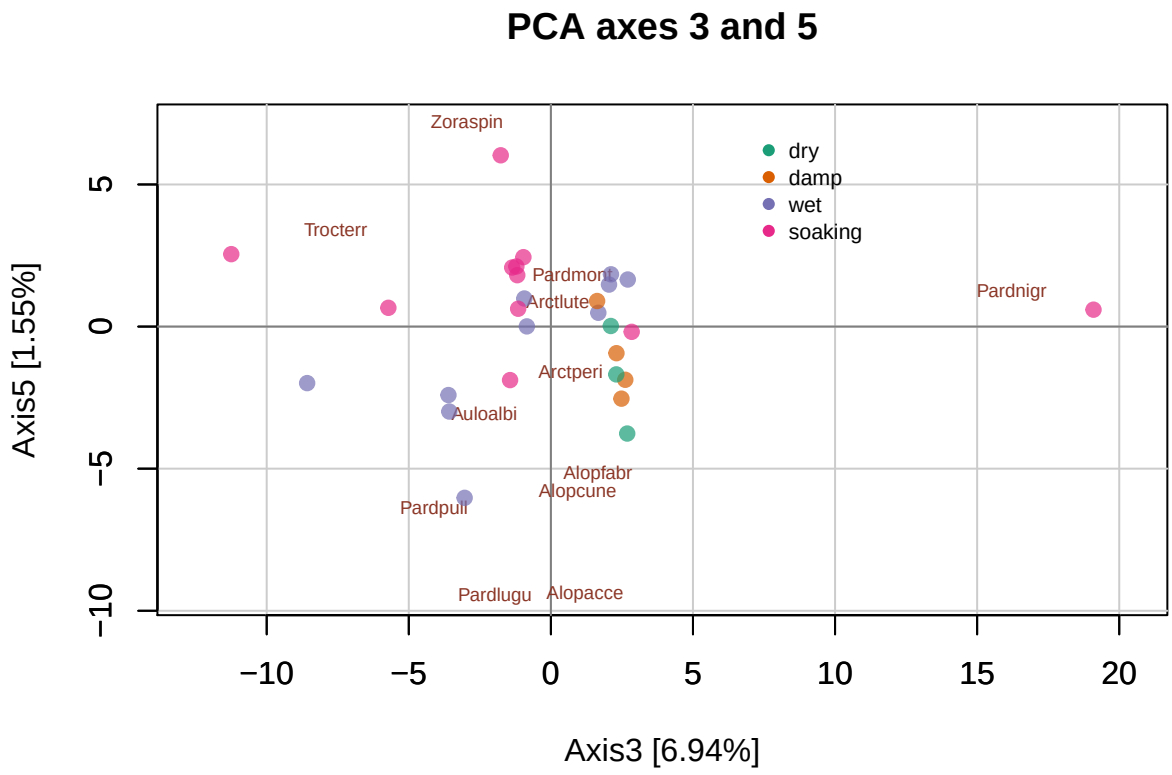


5.2 Different axes

Plot different axes

```
plot_ord(pca,
         row_group=row_group,
         axes=c(3,5),
         main='PCA axes 3 and 5',
         color_legend_position=c(0.6, 0))
```

```
## Rescaling standard scores by 5.487
```

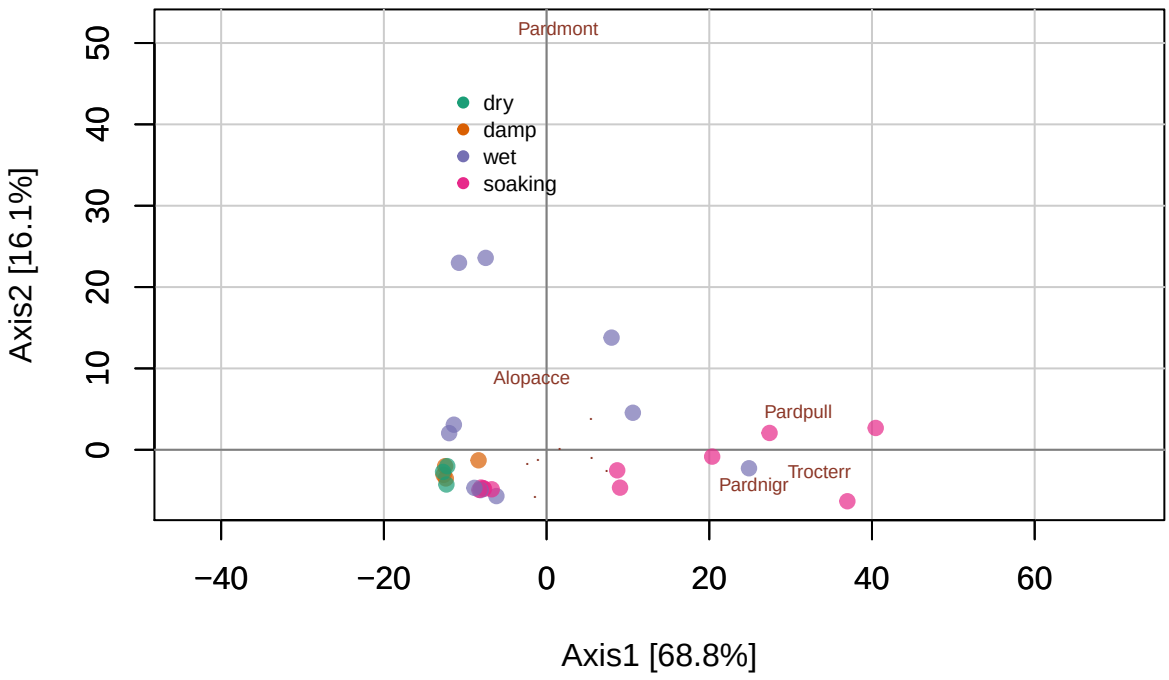


5.3 Limit column labels

Reduce clutter by only labeling the top columns (by plotted score)

```
plot_ord(pca,
         row_group=row_group,
         max_labels=5)
```

```
## Rescaling standard scores by 15.45
```



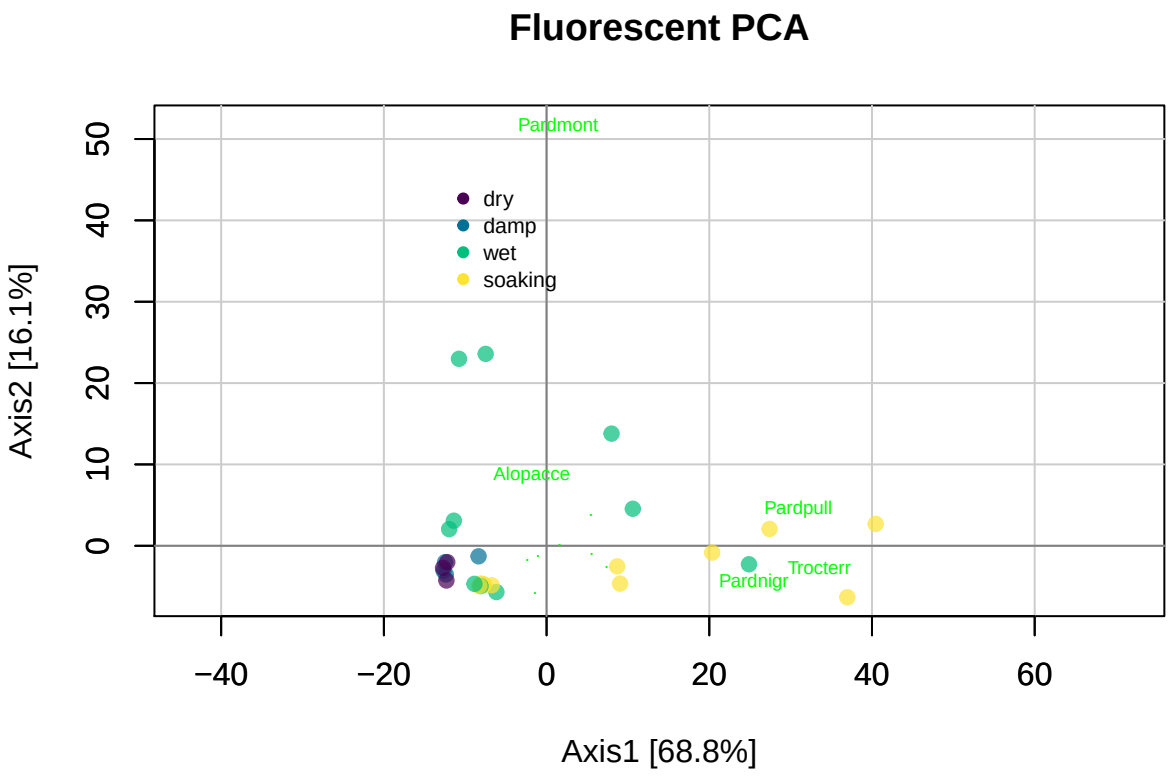
5.4 colors and shapes

Change the colors of text, groupings, arrows, and change shapes.

```
new_colors = hcl.colors(4, palette='Viridis')

plot_ord(pca,
         row_group=row_group,
         max_labels=5,
         discrete_scale=new_colors,
         col_text='green',
         main='Fluorescent PCA')
```

```
## Rescaling standard scores by 15.45
```



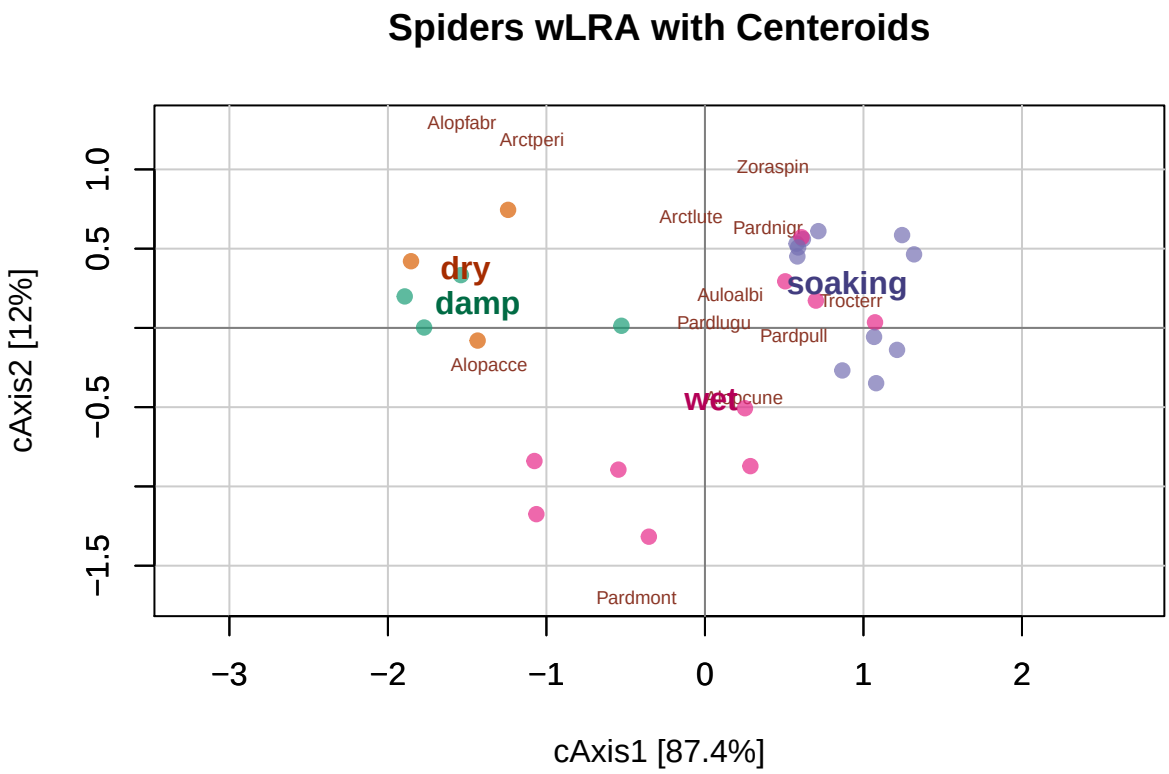
If you constrain an ordination with categories, the plot function will automatically recognize these and plot them as named centeroids:

```
X = envi[, 'soil.cat', drop=FALSE]
Y = CLR(closed_comm)

cat_cwLRA = pcwOrd(Y, X)

plot_ord(cat_cwLRA,
         main='Spiders wLRA with Centeroids')
```

```
## Rescaling standard scores by 0.8593
```

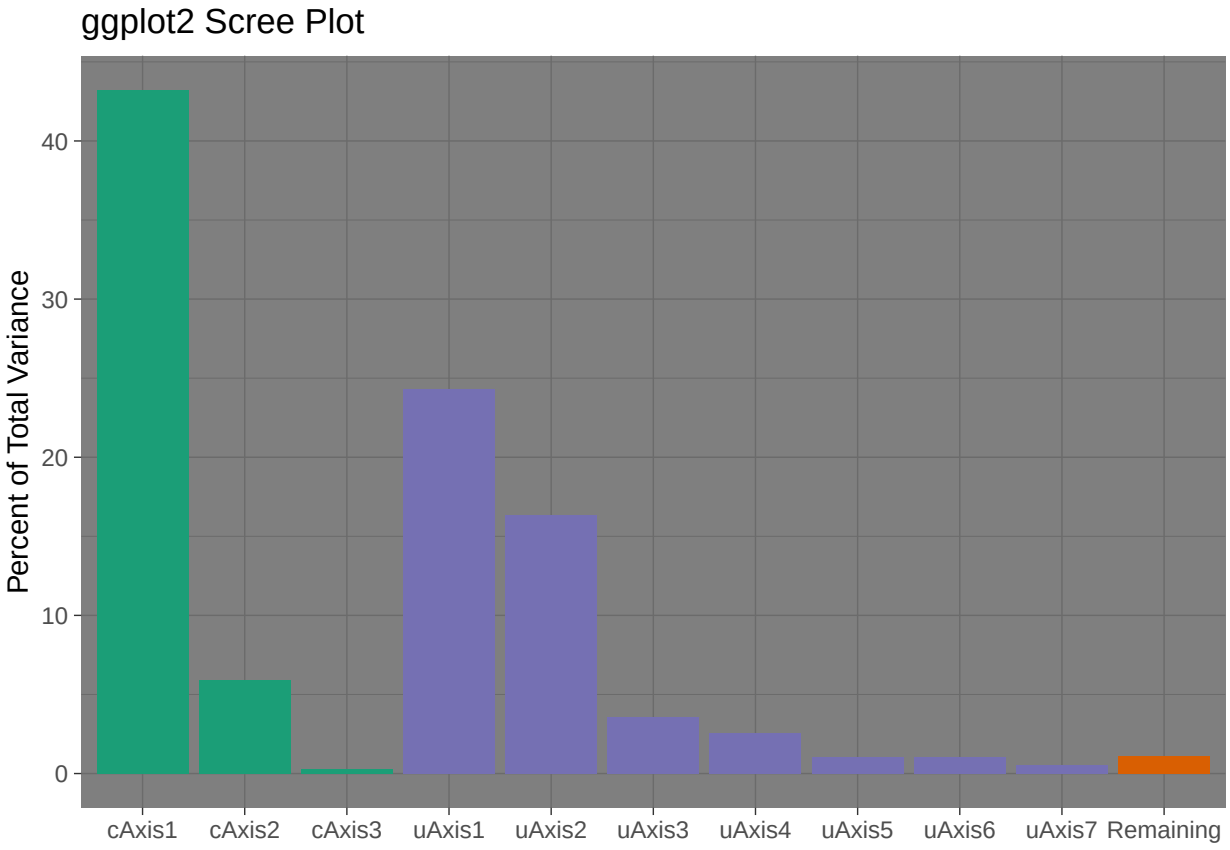


5.5 Screeplots as grobs

Screeplots can be produced as grobs (via ggplot2) for downstream manipulation, saving as R objects, and arranging with `gridExtra`. If you want to do this with ordinations, you're on your own!

```
ord_scree(cat_cwLRA, main='ggplot2 Scree Plot', as_grob=TRUE) +
  theme_dark()
```

```
## Loading required package: ggplot2
```


ggplot2 Scree Plot
Percent of Total Variance
40
30
20
10
0
cAxis1
cAxis2
cAxis3
uAxis1
uAxis2
uAxis3
uAxis4
uAxis5
uAxis6
uAxis7
Remaining