Correspondence regression: A tutorial

Koen Plevoets

May 8, 2017

1 By way of introduction: A bird's eye view of correspondence regression

Correspondence regression rests on the idea, described by Gilula and Haberman (1988), of modelling a multi-category response variable in terms of several (categorical) explanatory variables. Consider the built-in data set (in R) `HairEyeColor`, which gives the distribution of 592 students with respect to their hair color, eye color and sex:

```
HairEyeColor
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

A similar data table (with other subjects and without the `Sex` variable) was used by Fisher (1940), where he laid the foundations of the technique of *correspondence analysis* (together with Hirschfeld 1935). Although Fisher was primarily concerned with the association between hair color categories and eye color categories, one could see `Eye` color as a response variable and `Hair` color and `Sex` as

two explanatory variables. A geneticist, for instance, might well be interested in predicting the color of people's iris (i.e. their eye color) on the basis of their hair color and sex. Correspondence regression is meant for such an analysis.

The `HairEyeColor` data set is also contained in the **corregp** package, where it is reshaped into the data frame `HairEye` (and some of the labels have also been renamed):

```
library(corregp)
data(HairEye)
summary(HairEye)
##       Hair          Eye          Sex
##  Black  :108   Blue   :215   Female:313
##  Blond  :127   Brown_E:220   Male  :279
##  Brown_H:286   Green  : 64
##  Red    : 71   Hazel  : 93
ftable(HairEye, col.vars = "Eye")
##                Eye Blue Brown_E Green Hazel
## Hair    Sex
## Black   Female        9      36     2     5
##         Male         11      32     3    10
## Blond   Female       64       4     8     5
##         Male         30       3     8     5
## Brown_H Female       34      66    14    29
##         Male         50      53    15    25
## Red     Female        7      16     7     7
##         Male         10      10     7     7
```

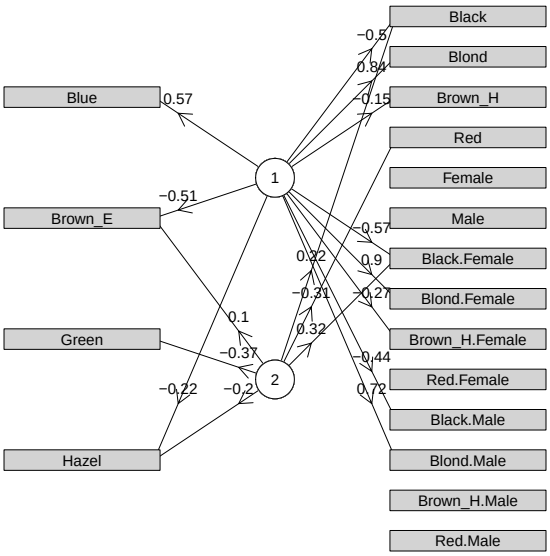
The 'flat' contingency table (produced with the `ftable()` function) nicely illustrates how the distribution of the `Eye` color categories can be studied in function of the (combination of) the `Hair` color categories and the `Sex` categories.

The package **corregp** has a single function for correspondence regression: the eponymous `corregp()`. The name of this function is a reference to the function `corresp()` from the package **MASS** (Venables and Ripley 2002) for simple correspondence analysis, and `corregp()` shares some of the computational features of `corresp()`. In line with the goal of regressing a response variable on explanatory variables, `corregp()` takes a typical R formula as its first argument: e.g. `Eye ~ Hair * Sex` performs a correspondence regression of the response variable `Eye` in function of the (combination of the) two explanatory variables `Hair` and `Sex` (more specifically, it performs a correspondence regression of `Eye` in function of the main effect of `Hair +` the main effect of `Sex +` plus the interaction between `Hair` and `Sex`, i.e. `Hair:Sex`). For more details on R formulas, see `help(formula)` or Section 2.1. Note that all variables will automatically be treated as categorical (i.e. R 'factors'): if you specify a numeric variable somewhere, then `corregp()` will convert it to a categorical/factor variable! The data frame containing the (categorical) variables can be specified as a second argument. The `corregp()` function contains many other arguments (see Section 2.1 for an overview, or read the function's help page: `help(corregp)`), but

one important one is the argument `b` which specifies the number of bootstrap replications (in fact, these are Monte Carlo simulations). This is relevant if one wishes to study the inferential properties of the results, such as confidence regions (the default value for `b` is `0`, which leaves the analysis exploratory). See Appendix 2 for more details on the inferential procedure. In the example below, we choose a random seed of `12345` (with the function `set.seed()`) in order to make the results reproducible (so, if you also copy-paste it, then you will obtain the same confidence regions as are printed in this tutorial):

```
set.seed(12345)
haireye.crg <- corregp(Eye ~ Hair * Sex, data = HairEye, b = 3000)
summary(haireye.crg)
## Summary of correspondence regression of Eye ~ Hair * Sex in HairEye
##
## Chi-squared: 150.0845
## Phi-squared: 0.2535211
## N: 592
##
##
## Eigenvalues:
##                  1          2          3    TOTAL
## value 130.6530753 16.7450400 2.68637037 150.0845
## %       0.8705302  0.1115708 0.01789905   1.0000
## cum_%   0.8705302  0.9821009 1.00000000   1.0000
```

The summary of the correspondence regression gives some overall statistics and it lists the distribution of the so-called 'eigenvalues'. The `Chi-squared` value is the same as Pearson's Chi-squared statistic of the above-mentioned 'flat' contingency table (which you can test yourself: type in
`chisq.test(ftable(HairEye, col.vars = "Eye"), correct = FALSE)`
and compare the value in the `X-squared` field of the output). The `Phi-squared` value is equal to the `Chi-squared` value divided by `N`, the total number of observations. Both the `Chi-squared` and the `Phi-squared` value express the dependence between the response variable (`Eye`) and the (combined) explanatory variables (`Hair` and `Sex`). The core idea behind correspondence regression is the same as behind correspondence analysis (or the *correlation models* of Gilula and Haberman 1988): these techniques assume that the response variable and the explanatory variables can be modelled in terms of underlying, latent axes which *explain* the observed dependencies. The underlying axes can also be thought of as *latent variables*, and correspondence analysis calls them 'principal axes'. For instance, the first two axes underlying the association between the `Eye` variable and the combination of the `Hair` and `Sex` variables can be illustrated in the following 'association graph' (the details behind association graphs will be explained later in this tutorial):



The eigenvalues in the summary indicate the 'explanatory power' of each principal or latent axis (correspondence analysis also speaks of 'principal inertias', which are the eigenvalues divided by `N`). They are given in three ways: the first row (`value`) shows the actual eigenvalues, the second row (`%`) shows the relative values, and the third row (`cum_%`) shows the cumulative relative values. One sees that the sum of the actual eigenvalues is equal to the `Chi-squared` value (the 'principal inertias' of correspondence analysis likewise sum to the `Phi-squared` value). The interpretation is straightforward: the latent axes *decompose* the observed association (between response variable and explanatory variables) into different sets.

Because we used the `summary()` function without any arguments, the reported eigenvalues are descriptive measures of the data set (for precise details on the computation of the latent axes, see standard textbooks on correspondence analysis such as Greenacre 2017, or see Appendix 1). However, since we have applied correspondence regression with bootstrapping (Monte Carlo simulations), we can compute *confidence intervals* for the eigenvalues. The confidence intervals are printed by setting the argument `add_ci` to `TRUE`:

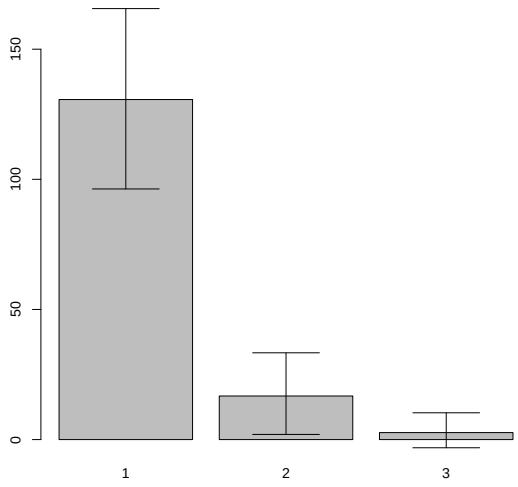
```
summary(haireye.crg, add_ci = TRUE)
## Summary of correspondence regression of Eye ~ Hair * Sex in HairEye
##
## Chi-squared: 150.0845
## Phi-squared: 0.2535211
## N: 592
##
```

```
##
## Eigenvalues:
##  Value:
##               1         2         3    TOTAL
##       130.65308 16.745040  2.686370 150.0845
## lower  96.30073  1.979968 -3.142215
## upper 165.60001 33.342428 10.294523
##
##  Percentage (%):
##               1          2           3 TOTAL
##       0.8705302 0.11157076  0.01789905     1
## lower 0.7614122 0.02415321 -0.01884798
## upper 0.9630913 0.20496508  0.06522083
##
##  Cumulative percentage (cum_%):
##               1         2         3 TOTAL
##       0.8705302 0.9821009 1.0000000     1
## lower 0.5728469 0.5722509 0.5663634
## upper 1.3599762 1.3510304 1.3613264
```

For the actual eigenvalues (in `Value`), the percentages, or the cumulative percentages, we see the observed measures together with a lower confidence bound (in `lower`) and an upper confidence bound (in `upper`). For example, the confidence interval for the first eigenvalue is $[96.30; 165.60]$, the confidence interval for the second eigenvalue is $[1.98; 33.34]$, the confidence interval for the first *relative* eigenvalue is $[0.76; 0.96]$, etc. By default, these are 95% confidence intervals, but you can specify the confidence level yourself with the argument `cl` (see `help(summary.corregp)`).

It is the philosophy of the **corregp** package that as many results as possible can be visualized. The eigenvalues can therefore be plotted in a 'scree plot' together with their confidence intervals. The **corregp** package has the function `screeplot` which also has the argument `add_ci`:

```
screeplot(haireye.crg, add_ci = TRUE)
```

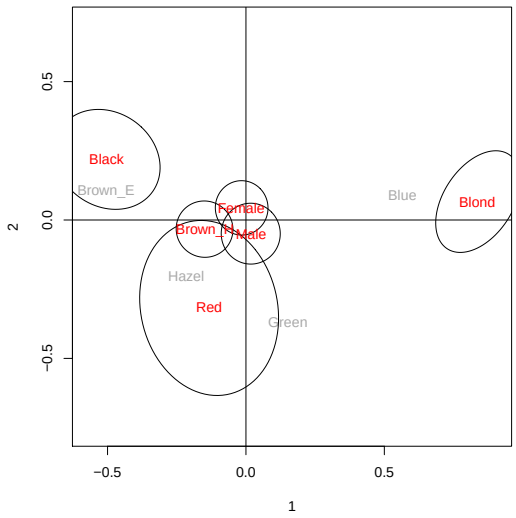


On the basis of the eigenvalues it can be determined which latent axes are 'important'. As mentioned, the eigenvalues measure the observed association between response and explanatory variables, and the results always sort the eigenvalues from the largest value to the smallest one. Hence, the first eigenvalues indicate important or informative axes, whereas the last eigenvalues indicate uninformative ones. Because any data set is always a random sample, these uninformative axes can be considered as reflecting the 'sampling noise' in the data. In practice, one inspects the eigenvalues for a certain cutoff point between the informative and the noisy axes. In a scree plot one typically looks for an 'elbow' among the eigenvalues: the first few eigenvalues generally exhibit a sharp decline with large differences between successive eigenvalues, but after a while the majority of the information (i.e. association) has been 'explained', so the differences between the later successive eigenvalues are not so large anymore. Such a point can be said to represent an elbow in the scree plot. The scree plot for the `HairEye` data contains only a few latent axes, so an elbow is not so easy to discern, but one could claim that there exists one at the second axis. That means that the first two latent axes are the informative ones. Because the eigenvalues are descriptive statistics of the data, such a statement is not based on statistical inference. However, if we have computed confidence intervals for the eigenvalues, then we can use them to back up our decision. In the output of the `summary()` function or in the scree plot, we see that the `lower` limit of the second eigenvalue is *lower* than the `upper` limit of the third eigenvalue. In other words, the confidence interval of the second eigenvalue overlaps with the confidence interval of the third eigenvalue, so the difference between the two eigenvalues can be regarded as not statistically significant (incidentally, the lower limit of the third eigenvalue is a negative value, which means that

0 lies within the confidence interval or that the third eigenvalue is not significantly different from 0). The conclusion is that the confidence intervals of the eigenvalues also indicate that the informative latent axes are the first two.

Once we have decided to retain two latent axes, we can inspect the results by visually plotting them in a two-dimensional coordinate space. Because correspondence regression is based on correspondence regression, the typical two-dimensional display is a 'biplot', which shows the categories of both the response variable and the explanatory variables. In the **corregp** package, the biplot is made with the generic function `plot()`, which has many arguments for customizing the outlook (see `help(plot.corregp)`):

```
plot(haireye.crg, x_ell = TRUE, xsub = c("Hair", "Sex"))
```



The distances in the plot are (inverse) reflections of the associations between the categories. For example, the eye color `Brown_E` is highly associated with the hair color `Black`, as are the eye color `Blue` and the hair color `Blond`. Similarly, the hair color `Red` is associated with both the eye colors `Hazel` and `Green`, just as the hair color `Brown_H` (in the center of the plot) appears to be indiscriminate between the eye colors `Brown_E` and `Hazel`. Other interpretations can be read off from the plot in the same vein. Because we asked for confidence regions for the explanatory variables (with the argument `x_ell`), the plot exhibits a two-dimensional 'confidence ellipse' for every category of `Hair` and `Sex` (the computation of the confidence ellipses is done by a call to the `cell()` function; see `help(cell.corregp)`). As a consequence, we are able to see that the difference between the hair colors `Black` and `Blond` (in the distribution of the eye colors) is statistically significant, both are significantly different from `Brown_H`

and Red, but the latter are not significantly different from each other. Similarly, there is no statistically significant difference between the sexes Female and Male (again, with respect to the distribution of the eye colors). A corollary of regressing a response variable on multiple explanatory variables is that the categories of *different* explanatory variables can also be compared with each other. For instance, we see that the sex category Female is not significantly different from the hair colors Brown_H and Red, but it is significantly different from Black or Blond (and the same holds for Male).

The biplot above only contains the main effects of Hair and Sex because of our use of the argument xsub (see help(plot) or see Section 2.2 for more details on ysub, xsub or other arguments of plot()). We have left the (eight) combination categories of the interaction variable Hair:Sex out of the biplot for the sake of illustration, since their inclusion would render the plot rather dense (they could be visualized with xsub = "Hair.Sex"). However, it is a common practice in the context of regression to first determine the relative importance of the explanatory variables (in explaining the variation in the response variable) before examining the effects. This is the *regression* aspect of correspondence regression: it is the analysis of how strongly associated each explanatory variable is to the response variable. It is particularly relevant if one has specified many different interactions in the formula of the correspondence regression and one wants to find out which are the important ones (so one could subsequently use the xsub argument to select only those, for instance). The **corregp** package has a function anova() which produces a so-called 'ANOVA table', i.e. for every predictor term in the correspondence regression it lists the (explained) association with the response variable which is not due to the other predictor terms. Association is measured by means of the Pearson Chi-squared statistic, and if one uses the anova() function without any specification of the number of latent axes, then the values in the output are directly related to the Pearson Chi-squared statistics of the cross table formed by each predictor term with the response variable (the specification of the number of latent axes is, of course, evident after the examination of the eigenvalues and/or the scree plot, but we first want to describe the ANOVA table):

```
anova(haireye.crg)
## ANOVA Table
## (Type III Tests)
##
##                X^2      Lower     Upper
## Hair     138.289842 106.825612 186.35043
## Sex        1.529824  -2.335166  11.51477
## Hair.Sex  10.264820   3.665810  32.88477
```

It can be easily verified that the value 138.289842 in (the first column X^2 of) the output is the Pearson Chi-squared statistic of the cross table of Hair and (the response variable) Eye, while 1.529824 is the Pearson Chi-squared

statistic of the cross table of `Sex` and `Eye`, and `10.264820` is the Pearson Chi-squared statistic of the cross table of `Hair:Sex` and `Eye` *minus* `138.289842` and `1.529824`, i.e. `150.0845 = 138.289842 + 1.529824 + 10.264820`. In other words, the value (`X^2`) for `Hair.Sex` (in the third row) expresses the amount of association that the interaction term `Hair:Sex` exhibits with the response `Eye` which *cannot* be explained by the main predictors `Hair` and `Sex` together. This is the typical way of analysing the 'contribution' of every predictor term to the explanation of the response variable in regression (the `X^2` values in the ANOVA table are in fact computed according to the 'additive' definition of interactions by Darroch 1974 and Kroonenberg and Anderson 2006).

However, the construction of an ANOVA table for correspondence regression usually involves the specification of the number of latent axes. This can be done with the argument `nf` (see `help(anova.corregp)` for the other arguments of the `anova()` function). From the discussion of the eigenvalues and the scree plot above we know that the first two latent axes are the informative ones for `haireye.crg`, so we build the ANOVA table for those two latent axes (naturally, the `X^2` values are somewhat reduced, since we have omitted the information on the third latent axis):

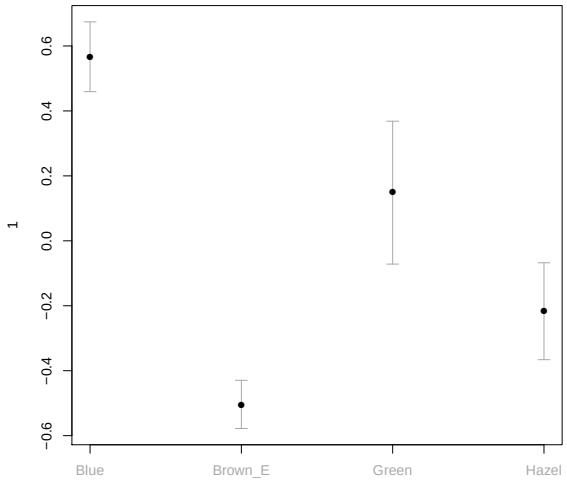
```
anova(haireye.crg, nf = 2)
## ANOVA Table
## (Type III Tests)
##
##                 X^2      Lower      Upper
## Hair     136.721942 103.269116 180.971757
## Sex        1.463949  -2.732508   9.726126
## Hair.Sex   9.212224   1.244894  26.972404
```

If the correspondence regression contains bootstrap replications/simulations (like `haireye.crg`), then the `anova()` function will also give confidence intervals for the `X^2` values, with which their statistical significance can be assessed. A predicter term is significant (and important) if `0` lies *outside* of its confidence interval. In the ANOVA table of the `HairEye` example, we see that the main effect of `Hair` as well as the interaction `Hair:Sex` are statistically significant, but the main effect of `Sex` is not. Apparently, the difference between the two sexes is not an important predictor for (the response variable) `Eye` color. This corroborates the result in the biplot above, where the individual categories `Female` and `Male` were found not to be significantly different from each other (it also means that the correspondence regression `haireye.crg` with one main effect and one interaction represents a so-called *non-hierarchical* model, but we will not pursue that issue further here). The presence of non-significant predictor terms does not necessarily imply the refitting of the correspondence regression, since the goodness of fit is determined with respect to the (number of) latent axes. Only when one wants to inspect new effects (or effects which were not included in the `formula` before) can one rerun the correspondence regression. However, the significant predictor terms in the ANOVA table bear on the subsequent steps

in the analysis, because they are the typical effects that one wants to visualize and study. That means that the biplot above should in fact have contained the effects of the interaction `Hair:Sex`, i.e. by setting `xsub = "Hair.Sex"`. In the remainder of this tutorial, we will keep using the main effects `Hair` and `Sex` for the sake of illustration, although the ANOVA table clearly points out that the interaction `Hair:Sex` is more informative.

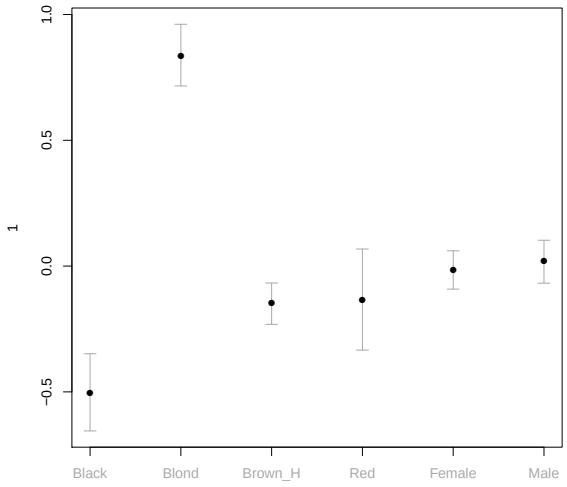
We continue this Introduction with some other plotting functionalities than the biplot. Although two-dimensional plots are customary for correspondence analysis, it is also possible to visualize the results of correspondence regression for one dimension, three dimensions or more. As a matter of fact, it is a general practice to use the (one-dimensional) confidence intervals for a single latent axis in order to determine whether the score of a particular category (on that latent axis) is significantly different from `0` or not. The plot of the confidence intervals of the categories on a certain latent axis (which you select) can be made with the function `ciplot()` (which itself calls the `cint()` function in order to compute the confidence intervals; see `help(cint.corregp)`). The `ciplot()` function has two important arguments among many other ones (see `help(ciplot.corregp)`): `parm` controls which categories are plotted and `axis` specifies the latent axis. The `parm` argument can be used very flexibly: the value `"y"` will plot all the categories of the response variable, the value `"x"` will plot all the categories of (all) the explanatory variables, a character vector with *names* of explanatory variables will plot only the categories of the specified explanatory variables, or finally, a character vector of *category names* (i.e. 'levels') will plot only the specified categories. For example, the plot of the confidence intervals of the response variable categories on the first latent axis is obtained as follows:

```
ciplot(haireye.crg, parm = "y", axis = 1)
```



This can then be compared to the confidence intervals of the categories of the explanatory variables (plots of confidence intervals for predictor variables are of course quite common in regression analysis). As mentioned above, we only illustrate the main categories of the `Hair` and `Sex` variables (so, not the categories of the interaction variable `Hair:Sex`):

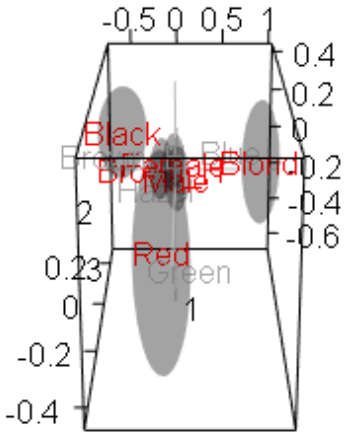
```
ciplot(haireye.crg, parm = c("Hair", "Sex"), axis = 1)
```



We see that the score of the hair color `Blond` (on the first latent axis) is significantly larger than `0`, the scores of both `Black` and `Brown_H` are significantly smaller than `0`, but the scores of the hair color `Red` as well as the two sexes `Female` and `Male` are *not* significantly different from `0` (similarly, the score of the eye color `Blue` is significantly larger than `0`, the scores of both `Black` and `Hazel` are significantly smaller than `0` while the score of `Green` is *not* significantly different from `0`). The same tests can be done for all the other latent axes as well as for the interaction `Hair:Sex`.

Because the **corregp** package imports the **rgl** package (Adler, Murdoch et al. 2017), it is also possible to make three-dimensional plots. This is of course not necessary for the `HairEye` data, where two latent axes are sufficient, but we will discuss plots of more than two dimensions for the sake of illustration. The function for 3D plots is `plot3d()`, which has special arguments for correspondence regression (see `help(plot3d.corregp)`). For example, we can ask for the three-dimensional confidence ellipsoids for the explanatory variables with the argument `x_ell` (which calls the `cell3d()` function for the 3D confidence ellipsoids; see `help(cell3d.corregp)`), and with the argument `xsub` we again visualize only the main categories of `Hair` and `Sex` (if you run the following code statement in R, then you can inspect the contents of the 3D plot by rotating it):

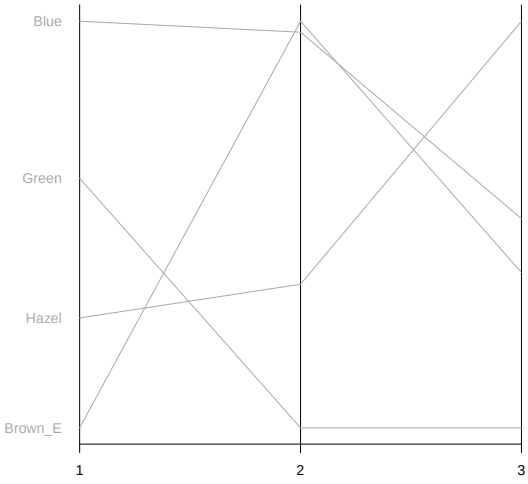
```
plot3d(haireye.crg, x_ell = TRUE, xsub = c("Hair", "Sex"))
```



If you happen to have a data set for which the eigenvalues point out that you need *more* than three latent axes, then you have three options (obviously, a single scatterplot of the results is no longer possible). A straightforward solution is to make multiple use of `ciplot()`, `plot()` or `plot3d()` in order to visualize

all the relevant latent axes. Both the `plot()` and `plot3d()` function have an `axes` argument with which you can select a combination of the latent axes (see the help pages of both functions). For instance, if you have an analysis with four important latent axes, then you can make four one-dimensional plots with `ciplot(, axis = 1)`, `ciplot(, axis = 2)`, `ciplot(, axis = 3)` and `ciplot(, axis = 4)`, or you can generate all six two-dimensional plots (or just a subset thereof) with `plot(, axes = c(1, 2))`, `plot(, axes = c(1, 3))`, `plot(, axes = c(1, 4))`, `plot(, axes = c(2, 3))`, `plot(, axes = c(2, 4))` and `plot(, axes = c(3, 4))`, or you can even create (a subset of) the four three-dimensional plots with `plot3d(, axes = c(1, 2, 3))`, `plot3d(, axes = c(1, 2, 4))`, `plot3d(, axes = c(1, 3, 4))` and `plot3d(, axes = c(2, 3, 4))`. An alternative is to make a 'parallel coordinate plot', in which the latent axes are displayed next to each other and the scores of individual categories are connected by a line. The function for a parallel coordinate plot in the **corregp** package is `pcplot()`, which again has many arguments for customization (see `help(pcplot.corregp)`). For example, a parallel coordinate plot of the `Eye` colors on the first three latent axes can be obtained as follows (just as the 3D plot above, this plot is not necessary for the `HairEye` data):

```
pcplot(haireye.crg, parm = "y", axes = 1:3)
```



Finally, you can visualize the results of a correspondence regression in an association graph, such as the one on pages 3–4 in this tutorial. Association graphs are *directed acyclic graphs*, in which the different latent axes are depicted as circles and the individual categories of both the response variable and explanatory variables are depicted as boxes. It is a convention to give the circles a

white color (for continuous variables) and the boxes a grey color (for discrete variables), but all colors can be changed manually. An association graph draws an arrow from a specific latent axis to a specific category *if and only if* the score of that category on that latent axis is significantly different from `0`, i.e. `0` does not lie within the confidence interval of that category on that latent axis (the confidence intervals are computed with the `cint()` function). In other words, if no arrow is drawn between a certain category and a certain latent axis, then this indicates that the score of that category on that latent axis is *not* significantly different from `0`, i.e. it is effectively `0`. The fact that the appearance of arrow between categories and latent axes is based on statistical significance entails, by the way, that association graphs can only be made if the correspondence regression contains bootstrap replicates/simulations! The (two) functions for visualizing association graphs in the **corregp** package are `agplot()` and `plotag()` (see either `help(agplot.corregp)` or `help(plotag.corregp)` for the help page), which in turn make use of the functionalities in the package **diagram** (Soetaert 2014). The association graph for the `HairEye` data above, for instance, can be obtained as follows:

```
agplot(haireye.crg, axes = 1:2)
```

For a general examination of the association between response variable and explanatory variables, the plots of the scores/coordinates are usually informative enough (especially the biplot). However, one can also look further into the relations between the latent axes and the individual categories. That can be useful for finding an *interpretation* for the latent axes. There are two measures: the *contributions of the points to the axes* express how well each category represents (the inertia corresponding to) a certain latent axis, while the *contributions of the axes to the points* express how well each latent axis reflects a certain category. The difference between both measures is essentially that the former contributions (i.e. of the points to the axes) sum to 100% for each latent axis, whereas the latter contributions (i.e. of the axes to the points) sum to 100% for each individual category. The former contributions are sometimes also called the 'absolute contributions', and the latter contributions are sometimes referred to as the 'squared correlations'. Both contributions can be consulted with the `summary()` function by specifying the argument `contrib`. That argument can have a plethora of possible values: the contributions of the points to the axes are given by `"p_a"`, `"pts_axs"`, `"pts2axs"`, `"ptstoaxs"`, `"pts_to_axs"`, `"pnts_axes"`, `"pnts2axes"`, `"pntstoaxes"` or `"pnts_to_axes"`, the contributions of the axes to the points are given by `"a_p"`, `"axs_pts"`, `"axs2pts"`, `"axstopts"`, `"axs_to_pts"`, `"axes_pnts"`, `"axes2pnts"`, `"axestopnts"` or `"axes_to_pnts"`. Last but not least, the `contrib` argument can also have the value `"both"` or `"b"`. In the following example, we specify the first two latent axes (with the argument `nf`), so the contributions of the axes to the points (`"axes2pnts"`) are of course less than 100%:

```
summary(haireye.crg, parm = "y", contrib = "axes2pnts", nf = 2)
## Summary of correspondence regression of Eye ~ Hair * Sex in HairEye
##
## Chi-squared: 150.0845
## Phi-squared: 0.2535211
## N: 592
##
##
## Eigenvalues:
##                 1          2       TOTAL
## value 130.6530753 16.7450400 147.3981153
## %       0.8705302  0.1115708   0.9821009
## cum_%   0.8705302  0.9821009   0.9821009
##
##
## Contributions:
##
## Y (Eye):
##  Axes to points (Squared correlations):
##                 1          2     TOTAL
## Blue    0.9744458 0.02539499 0.9998408
## Brown_E 0.9568324 0.04059033 0.9974227
## Green   0.1315237 0.78261334 0.9141370
## Hazel   0.4463229 0.39276850 0.8390914
```

```
summary(haireye.crg, parm = "x", contrib = "pts_axs", nf = 2)
## Summary of correspondence regression of Eye ~ Hair * Sex in HairEye
##
## Chi-squared: 150.0845
## Phi-squared: 0.2535211
## N: 592
##
##
## Eigenvalues:
##                 1          2       TOTAL
## value 130.6530753 16.7450400 147.3981153
## %       0.8705302  0.1115708   0.9821009
## cum_%   0.8705302  0.9821009   0.9821009
##
##
## Contributions:
##
## X:
##  Points to axes (Absolute contributions):
##
##    Hair:
##                   1          2
## Black   0.209316873 0.31484902
## Blond   0.678929483 0.03082431
```

```
## Brown_H 0.048040842 0.01930658
## Red     0.009571926 0.41988166
## TOTAL   0.945859124 0.78486157
##
##    Sex:
##                    1          2
## Female 0.0006555480 0.03608747
## Male   0.0007354356 0.04048523
## TOTAL  0.0013909836 0.07657270
##
##    Hair.Sex:
##                             1           2
## Black.Female   1.284378e-01 0.321689624
## Blond.Female   5.033496e-01 0.135983679
## Brown_H.Female 8.091262e-02 0.017295774
## Red.Female     1.963481e-02 0.165594391
## Black.Male     8.410841e-02 0.054144972
## Blond.Male     1.828826e-01 0.039051095
## Brown_H.Male   6.512204e-04 0.004223557
## Red.Male       2.299285e-05 0.262016907
## TOTAL          1.000000e+00 1.000000000
```

In summary, correspondence regression typically involves the following steps:

1. Perform a correspondence regression with the `corregp()` function.
2. Check the eigenvalues in order to determine the number of important latent axes. You can use either `summary()` or `screeplot()`.
3. Build an ANOVA table with `anova()` in order to discern the important predictor terms among the explanatory variables.
4. Visualize the results with `ciplot()`, `plot()`, `plot3d()`, `agplot()` (or `plotag()`) or `pcplot()`. In case you need more than two latent axes, you can use `ciplot()` or `plot()` (and even `plot3d()`) several times and make different selections of the latent axes.
5. If you want more information on the interpretation of the latent axes, then you can inspect the *contributions of the point to the axes* and/or the *contributions of the axes to the points* by specifying the `contrib` argument of `summary()`.

2 Functions and methods in the `corregp` package

2.1 The function `corregp()`

Because `corregp()` is the central function for correspondence regression, we will now explain each of its arguments (see also `help(corregp)`):

`formula` This should be a typical R formula with a response variable on the left-hand side and explanatory variables on the right-hand side (separated by a tilde). See Section *11.1 Defining statistical models; formulae* of the standard R manual *An Introduction to R* for an elaborate description. Examples for correspondence regression are (for convenience' sake, we will denote the response variable as `Y` and the explanatory variables as `X1`, `X2` etc.):

- `Y ~ X1` Correspondence regression of `Y` in function of `X1`. This is equivalent to a *simple correspondence analysis* of the two variables `Y` and `X1`.
- `Y ~ X1 + X2` Correspondence regression of `Y` in function of both `X1` and `X2`. The output will only contain the results for the main categories of `X1` and the main categories of `X2` (next to the main categories of `Y`).
- `Y ~ X1 + X2 + X1:X2` Correspondence regression of `Y` in function of the interaction between `X1` and `X2`. In other words, the output will contain results for the main categories of `X1`, the main categories of `X2` and the combined categories of the interaction `X1:X2`.
- `Y ~ X1 * X2` The same as `Y ~ X1 + X2 + X1:X2`.
- `Y ~ (X1 + X2 + X3)^2` Correspondence regression of `Y` in function of all the two-way interactions between `X1`, `X2` and `X3`. This is equivalent to `Y ~ X1 + X2 + X3 + X1:X3 + X1:X2 + X2:X3`.
- `Y ~ (X1 + X2 + X3)^2 - X2:X3` Correspondence regression of `Y` in function of all the two-way interactions between `X1`, `X2` and `X3` *except* the one between `X2` and `X3`. This is equivalent to `Y ~ X1 + X2 + X3 + X1:X2 + X1:X3`.
- `Y ~ X1 * X2 * X3 - X1:X2:X3` Correspondence regression of `Y` in function of all the *two-way* interactions between `X1`, `X2` and `X3`. The term `X1 * X2 * X3` denotes all the possible combinations of `X1`, `X2` and `X3`, but the three-way interaction `X1:X2:X3` is excluded. In other words, this is equivalent to `Y ~ X1 + X2 + X3 + X1:X2 + X1:X3 + X2:X3` (hence, also to `Y ~ (X1 + X2 + X3)^2`).
- `Y ~ X1/X2` Correspondence regression of `Y` in function of `X2` *nested with* `X1`. This is just the same as `Y ~ X1 + X1:X2` (or as `Y ~ X1*X2 - X2`).
- `Y ~ -1 + X1 * X2` The same as `Y ~ X1 * X2`. Intercepts are not part of correspondence regression in the first place, so their exclusion in the formula does not change anything.
- `Y ~ 0 + X1 * X2` The same as `Y ~ -1 + X1 * X2`, so also as `Y ~ X1 * X2`.
- ...

`data` This should be the data frame containing all the variables in `formula` as columns. IMPORTANT to remember is that any numeric or logical variable will be converted to a factor (i.e. a categorical variable).

`part` This can be a (character) vector of conditional variables for both the response variable and the explanatory variables: if specified, then a correspondence regression of the response variable and the explanatory variables will be performed *given* these variables. More specifically, if we denote such conditional variables as `Z1`, `Z2` etc., then correspondence regression of `Y ~ X1 + X2 +...` will basically amount to an analysis of `Y:(Z1:Z2:...) ~ X1:(Z1:Z2:...) + X2:(Z1:Z2:...) + ...`. Note that `Z1`, `Z2` etc. have to be columns in `data`, and the correct notation of the argument is `part = c("Z1", "Z2", ...)`, i.e. **with** quotation marks. A possible example of a conditional variable is a *grouping factor* for the categories of the response (`Y`) variable (which is relevant for so-called *lectometric* analyses in linguistics).

`b` This can be set equal to the number of bootstrap replications (Monte Carlo simulations). Those are new samples which are generated by resampling the observed data set (with replication). The new, replicated/simulated samples lead to to new values for both the eigenvalues and the scores on the latent axes. These new values can be used (by the functions `cint()`, `cell()` and `cell3d()`) to construct confidence regions. Usually, the number of replications/simulations is chosen to be quite large (e.g. 3000). If set to `0`, then no replicate samples are generated, so no confidence regions can be computed.

`xep` This argument stands for 'x separate'. By default, the results for all predictor terms in `formula` are collected as separate components in a list. This also admits the construction of an ANOVA table for the predictor terms. However, if you want the results of the explanatory variables collected in one overall matrix, then you can set this argument to `FALSE`.

`std` This argument specifies whether to standardize the latent axes or not. The unstandardized scores on the latent axes are also called the 'principal coordinates', and the variance of each latent axis is the corresponding eigenvalue. The standardized scores (i.e. with variance `1`) are also called the 'standard coordinates'.

`rel` By default, correspondence analysis (and correspondence regression) computes scores for the row profiles and the column profiles, i.e. the rows of a frequency table divided by their row totals and the columns of the table divided by their column totals. You can set this argument to `FALSE` if you want to obtain scores for the Pearson residuals ($\frac{O-E}{\sqrt{E}}$) instead. Leave this argument untouched unless you know what you are doing.

`phi` This argument specifies whether the eigenvalues in the output should sum to the *Chi-squared* value or to the *Phi-squared* value, which is the

Chi-squared value divided by the number of observations (see Appendix 1). In accordance with the `corresp()` function in the package **MASS**, the default for this argument is the first option.

`chr` If the `formula` contains some interaction terms, such as `X1:X2`, then the output will contain results for the combination of the categories/levels of `X1` with the categories/levels of `X2`. The `chr` argument specifies which character string will be used as the connector (or 'separator') for the combinations. The default for this argument is to combine the categories/levels with a single dot.

The output of the `corregp()` function is a list, the components of which are described in the `Value` section of the help file (`help(corregp)`). Each component can be accessed individually:

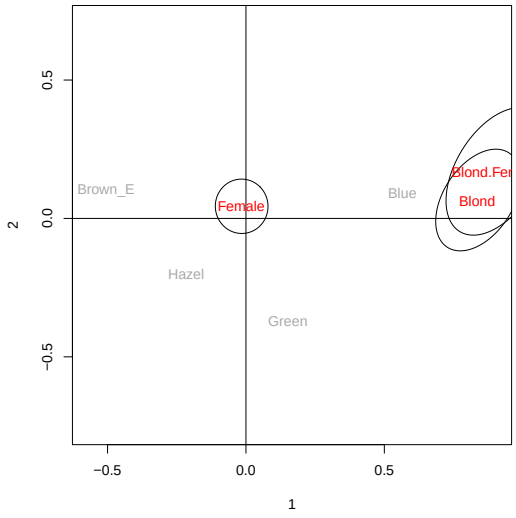
```
is.list(haireye.crg)
## [1] TRUE
names(haireye.crg)
## [1] "eigen" "y"     "x"     "freq"  "conf"  "aux"
haireye.crg$y
##                  1           2            3             4
## Blue     0.5652324  0.09124777  0.007224548 -2.650080e-17
## Brown_E -0.5052264  0.10405893 -0.026220924  2.245537e-17
## Green    0.1516430 -0.36990813 -0.122524570  6.938894e-18
## Hazel   -0.2159174 -0.20254960  0.129644064  1.439060e-17
```

2.2 The plotting functions

The plotting functions `ciplot()`, `plot()`, `plot3d()`, `agplot()` (`plotag()`) and `pcplot()` have many more arguments than the ones described in the Introduction. The majority of these involve settings for the usual graphical parameters such as color, font size, font type, line width, line type, main title of the plot, subtitle of the plot, labels for the axes and/or limits for the axes (and so on). See the help pages of the functions for a full overview (`help(ciplot)`, `help(plot.corregp)`, `help(plot3d.corregp)`, `help(agplot)` ,`help(pcplot)`). Probably the best way to master these arguments is by trial.

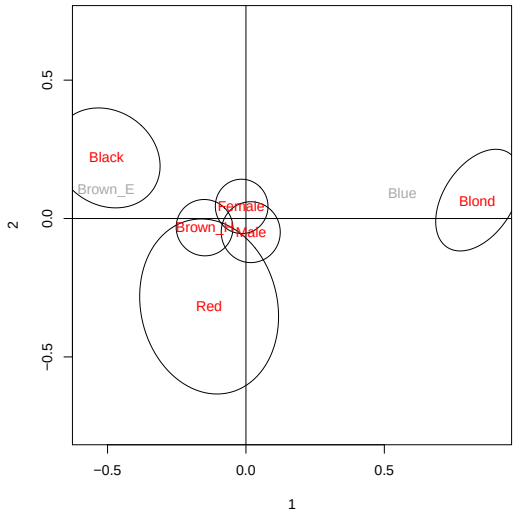
The functions `plot()`, `plot3d()` and `agplot()` (or `plotag()`) contain two arguments `ysub` and `xsub` which require some further clarification. In the examples in the Introduction, `xsub` was used with the names of explanatory variables. That automatically selected the categories (i.e. 'levels') belonging to those variables. However, both `ysub` and `xsub` can also be specified with indices for the individual categories themselves. The following code, for instance, plots only the main Hair category `Blond`, the main Sex category `Female` and the interaction category `Blond.Female` from the `X` results (and `ysub` would work in the same way):

```
plot(haireye.crg, x_ell = TRUE, xsub = c("Blond", "Female", "Blond.Female"))
```



Both arguments can also be given *numeric* indices, but there is a difference between `ysub` and `xsub`. Numeric indices for `ysub` simply select the corresponding rows from the `Y` table in the same way as character names do. Numeric indices for `xsub`, however, depend on whether the `corregp()` output was generated with the argument `xep` being `TRUE` or `FALSE`. If `xep = TRUE`, then the results for `X` form a list themselves and the numeric indices select the corresponding components from that list, i.e. all the categories of the selected predictor term(s). If `xep = FALSE`, then the results for `X` are all contained in one overall matrix, and the numeric indices (again) select the corresponding rows. In other words, there is no way of selecting individual categories with numeric indices if `xep = TRUE` (one has to use character values or rerun `corregp()` with `xep = FALSE`). The following code illustrates the use of numeric indices by plotting the `Blue` and `Brown_E` Eye colors as well as all the categories of both the main predictor `Hair` and the main predictor `Sex`:

```
plot(haireye.crg, x_ell = TRUE, xsub = c(1, 2), ysub = c(1, 2))
```



2.3 The functions for confidence regions

Although confidence regions in the `corregp` package are primarily computed for visualization, there are also functions which give the actual numeric output. These are `cint()` for (one-dimensional) confidence intervals, `cell()` for (two-dimensional) confidence ellipses and `cell3d()` for (three-dimensional) confidence ellipsoids. Normally, users do not have to call these functions directly (as they are called by `ciplot()`, `plot()` and `plot3d()`, respectively), but one can do so if one has a certain use for them (see `help(cint)`, `help(cell)` or `help(cell3d)`). For instance, the confidence intervals of the four `Eye` colors on the first latent axis are:

```
cint(haireye.crg, parm = "y", axis = 1)
##               Lower       Upper
## Blue     0.45935041  0.67427396
## Brown_E -0.57795504 -0.42951504
## Green   -0.07172607  0.36835832
## Hazel   -0.36603940 -0.06768381
```

All three functions contain the argument `cl` which specifies the confidence level for the confidence regions (the default value is the conventional `0.95`). This is the percentage of areas which would contain the true population value (i.e. of a certain score) if the sample were repeated. Because `corregp()` works with bootstrap replications (simulations), this means that the confidence level `cl` specifies the percentage of the `b` replicate values (for each score) used to construct the confidence region.

The `cint()` function also has the argument `nq`, which specifies whether one wants to construct the confidence interval using the normal distribution or not. The use of the normal distribution (with `nq = TRUE`) means that the mean and the standard deviation of the `b` replicate values are computed on the basis of which a confidence interval is constructed under the normal distribution (i.e. by means of the function `qnorm()`). If one does not use the normal distribution (with `nq = FALSE`), then the confidence interval is obtained from the `b` replicate values themselves by means of the `quantile()` function (i.e. by choosing two actual replicate values as the lower and upper limit which together represent the confidence level `cl`). The computation of bootstrap confidence intervals is usually done in the second 'non-parametric' way, but the first option is available as the (one-dimensional) counterpart of two-dimensional confidence ellipses (made with `cell()`), because the latter are *always* constructed by means of the bivariate normal distribution. It probably needs no clarification that normal confidence intervals and non-parametric confidence intervals can sometimes be quite different. For completeness' sake, it should be pointed out that both the `summary()` function and the `screeplot()` function also contain the arguments `cl` and `nq` for the confidence intervals of the *eigenvalues*, where they have the same meaning. All confidence intervals in the **corregp** package are computed with the function `ci()` (see `help(ci)`).

The `cell()` function does not have an argument `nq`, but it does have an argument `np`. That argument specifies the number of points to represent an ellipsis. These points are connected by lines in the visualization to form the ellipsis. The representation of ellipses by a series of `np` points is due to the fact that `corregp()` makes use of the **ellipse** package (Murdoch and Chow 2013) to compute the confidence ellipses (which is inspired by R code on www.carme-n.org). The default number of points is `100`, which usually gives good results. One can increase this value if one wishes a better resolution, but that will lead to a longer computation time.

Finally, the `cell3d()` function makes use of the `ellipse3d()` function from the **rgl** package (which itself creates a so-called `mesh3d` object). See the help pages of the **rgl** package for further information. The `cell3d()` function has no additional arguments.

2.4 The functions for extracting coefficients, fitted values or residuals

The **corregp** package also has functions for extracting coefficients, fitted values and/or residuals from a correspondence regression. These are conventially named `coefficients()`, `fitted.values()` and `residuals()` with their customary abbreviations `coef()`, `fitted()` and `resid()`. The coefficients of correspondence regression are essentially the coordinate scores of the categories on the latent axes, so the function `coefficients()` (or `coef()`) prints them in a

matrix or a vector. It is used with the argument `parm` for the selection of the categories (i.e. `"y"` for all the `Y` categories/levels, `"x"` for all the `X` categories/levels, a vector of any variable/term names in `X`, a vector of any category/level names in `X` or a vector of any category/level names in `Y`) and the argument `axes` for the selection of the latent axes (see `help(coefficients.corregp)`). The function `fitted()` (or `fitted.values()`) makes use of the coordinate scores in correspondence regression to compute predicted/expected frequencies for every cell in the cross table of `X` by `Y`. It is used with the argument `parm` (in the same way as before) and the argument `nf` for the selection of the number of latent axes (see `help(fitted.corregp)`). The function `residuals()` (or `resid()`) is the complement to `fitted()` in that it computes the difference between observed frequencies and predicted frequencies for every cell in the cross table of `X` and `Y` (in other words, for a certain number of latent axes, the sum of the fitted frequencies and the residuals is equal to the observed frequencies — except when conditional variables have been specified in `part`, because the computation does not take the associations with these variables into account). The function is also used with the arguments `parm` and `nf` (see `help(residuals.corregp)`). Examples for these three functions are:

```
coef(haireye.crg, parm = c("Hair", "Sex"), axes = 1:2)
##                    1           2
## Black   -0.50321092  0.22094409
## Blond    0.83573827  0.06375113
## Brown_H -0.14814333 -0.03362116
## Red     -0.13271815 -0.31468597
## Female  -0.01654207  0.04393890
## Male     0.01855795 -0.04929346
```

```
fitted(haireye.crg, parm = c("Hair", "Sex"), nf = 2)
##              Blue    Brown_E     Green    Hazel
## Black    20.17722  67.341848  4.105338 16.37560
## Blond    94.09737   6.638403 15.508462 10.75577
## Brown_H  83.45987 121.005932 31.726772 49.80742
## Red      17.26554  25.013817 12.659428 16.06121
## Female  114.12142 121.549078 30.387038 46.94247
## Male    100.87858  98.450922 33.612962 46.05753
```

```
resid(haireye.crg, parm = c("Hair", "Sex"), nf = 2)
##                Blue    Brown_E      Green      Hazel
## Black   -0.17721681  0.6581523  0.8946619 -1.3755974
## Blond   -0.09736508  0.3615970  0.4915382 -0.7557700
## Brown_H  0.54012560 -2.0059324 -2.7267717  4.1925784
## Red     -0.26554371  0.9861831  1.3405716 -2.0612110
## Female  -0.12141708  0.4509219  0.6129623 -0.9424671
## Male     0.12141708 -0.4509219 -0.6129623  0.9424671
```

Appendix 1: The computation of correspondence regression

Correspondence regression starts from a frequency table formed by crossing the response variable (`Y`) with all the possible combinations of the explanatory variables (`X`). The latter is the same as the highest-order interaction of all the explanatory variables, i.e. if the `formula` is specified as e.g. `Y ~ X1 + X2 + X3`, then correspondence regression crosstabulates `Y` with `X1:X2:X3` (in the usual R notation). If conditional variables (`Z1`, `Z2`, ...) are specified in the argument `part`, then correspondence regression constructs a three-way table by crossing `Y` and `X` for every possible combination of the conditional variables (i.e. the joint distribution of the conditional variables). Correspondence regression then computes the *Pearson residuals* $\frac{O-E}{\sqrt{E}}$ of this table, where E is calculated according to the usual formula of *conditional independence* of `Y` and `X` given `Z` (if no conditional variable is specified in `part`, then E is simply calculated as the *mutual* independence of `Y` and `X`). The three-way table is subsequently aggregated over the conditional variables with weights $\frac{n_{+jk}}{n_{+j+}}$ for every `Y` level j and `Z` level k and weights $\frac{n_{i+k}}{n_{i++}}$ for every `X` level i and `Z` level k. The resulting matrix (of `Y` versus `X`) measures the same association as the three-way correspondence analysis in Section 3.2 of Van der Heijden et al. (1989) (who make use of Escofier's 1984 generalized correspondence analysis). If `phi = FALSE`, then we denote this matrix of (aggregated) Pearson residuals as D. Otherwise, if `phi = TRUE`, then we let D be the matrix of (aggregated) Pearson residuals divided by $\sqrt{N}$ (i.e. divided by the square root of the total number of observations).

Just as correspondence analysis, correspondence regression computes the Singular Value Decomposition of D:

$$\mathsf{D} = \mathsf{USV}^{\mathsf{T}}$$

The matrix U contains scores/coordinates for the explanatory (`X`) categories and the matrix V contains scores/coordinates for the response (`Y`) categories, which both depend on the value of the argument `rel`. The diagonal matrix S contains the so-called *singular values*, which are the square roots of the eigenvalues. In other words, the eigenvalues are the diagonal values of S^2.

If `rel = FALSE`, then the matrices U and V contain the *standardized* scores/coordinates of the `X` and `Y` categories, respectively. The matrix U has a row for every possible combination in `X`, so the more general categories (i.e. the main categories and/or the lower-order interactions) are obtained by aggregating the appropriate rows of U. If `rel = TRUE`, then the *standardized* scores/coordinates are computed by dividing the rows of U and V by the *square roots of the corresponding total frequencies* of the `X` and `Y` categories, respectively. More specifically, let $\mathbf{r}$ be (the vector of) the total frequencies of all `X` categories and let $\mathbf{c}$ be (the vector of) the total frequencies of the `Y` categories (they are, of course, also

the row and column totals of the original cross table of `X` and `Y`). Then the standardized scores/coordinates of the `X` categories are obtained with $\text{diag}\left(\frac{1}{\sqrt{\mathbf{r}}}\right) * \mathsf{U}$ and the standardized scores/coordinates of the `Y` categories are obtained with $\text{diag}\left(\frac{1}{\sqrt{\mathbf{c}}}\right) * \mathsf{V}$. Again, the U matrix only contains rows for every possible combination of the `X` variables, so the score/coordinate of a more general category can be obtained by aggregating the appropriate rows of U as well as summing the corresponding totals in $\mathbf{r}$ and multiplying these two. In other words, if the aggregated rows in U for a certain lower-order category be denoted as U_+ and the corresponding sum of the totals be denoted as $\mathbf{r}_+$, then the standardized score/coordinate of the lower-order category is $\text{diag}\left(\frac{1}{\sqrt{\mathbf{r}_+}}\right) * \mathsf{U}_+$.

The scores/coordinates which are actually outputted finally also depend on the argument `std` (as was explained in Sextion 2.1). If `std = TRUE`, then the scores/coordinates in the output for both the `X` and the `Y` categories are the *standardized* scores/coordinates, which were just discussed. If `std = FALSE`, then the output contains the so-called *principal* scores/coordinates, which are computed by multiplying the standardized scores/coordinates of both `X` and `Y` by the matrix S.

The contributions of the points to the axes as well as the contributions from the axes to the points can also be obtained from the matrices U, S and V. The contributions of the points to the axes (i.e. the 'absolute contributions') for the `X` categories are in the columns of the matrix U^2, which each sum to 1. In other words, for each latent axis, the contributions of the `X` categories to the axis are shown in the columns of U^2. Likewise, the contributions of the points to the axes of the `Y` categories are in the columns of V^2. The contributions of the axes to the points (i.e. the 'squared correlations') involve some matrix multiplication, so that the contributions can be read from the rows of the resulting matrices. For the `X` categories, the contributions of the axes to the points are in the rows of:

$$(\mathsf{US})^2 * \text{diag}\left(\frac{1}{\sum_k (\mathsf{US})_{ik}^2}\right)$$

For the `Y` categories, the contributions of the axes to the points can be obtained in the same fashion as the rows of:

$$(\mathsf{VS})^2 * \text{diag}\left(\frac{1}{\sum_k (\mathsf{VS})_{jk}^2}\right)$$

Appendix 2: The computation of confidence regions

As has been mentioned multiple times in this tutorial, the `corregp()` function is able to compute confidence regions for its results by means of a bootstrap

procedure; in particular, this is Monte Carlo simulation. Bootstrapping/ Simulation involves generating many (i.e. `b`) new, replicated samples by *sampling the observed data set with replacement* (the replicated samples always have the same size `N` as the observed data set). Because the data are arranged in a frequency table (by crossing the response variable `Y` with all combinations of the explanatory variables in `X`, sometimes extended to a three-way frequency table in case of conditional variables in `part`), the resampling is more specifically done with *multinomial sampling*. Any frequency table can be thought of as a multinomial variable, and the division of the cell counts by the total sample size `N` gives natural estimates of the individual cell probabilities. The estimated cell probabilities and the sample total `N` are passed as arguments to the built-in R function `rmultinom()`, which generates `b` new random multinomial data samples. These are the replications/simulations of the observed data set.

Each of these replicated tables lead to bootstrap/simulated replicates of the matrices U, S and V. This is done, in particular, by means of the *partial bootstrap* procedure, which is outlined in Alvarez et al. (2002; 2004; 2006) and Lebart (2004). The partial bootstrap makes use of the fact that the formula for the Singular Value Decomposition (see Appendix 1 above) can be rewritten into formulas for U, S and V themselves. By substituting a specific bootstrap/simulated table for the observed table in the application of these formulas, the bootstrap/simulated replicates of the U, S and V matrices can be obtained (in other words, these are computed in the same way as supplementary points in correspondence analysis). More specifically, denote the Pearson residuals of a particular bootstrap/simulated table as D^*. Then, the bootstrap/simulated replicate of the matrix U, which will be denoted as U^*, can be derived as follows:

$$\mathsf{U}^* = \mathsf{D}^*\mathsf{V}\mathsf{S}^{-1}$$

Similarly, the bootstrap/simulated replicate of V, denoted as V^*, is obtained as:

$$\mathsf{V}^* = \mathsf{D}^{*\mathsf{T}}\mathsf{U}\mathsf{S}^{-1}$$

Finally, the bootstrap/simulated replicate of S (hence, of the eigenvalues), denoted as S^*, can be computed as:

$$\mathsf{S}^* = \mathsf{U}^{\mathsf{T}}\mathsf{D}^*\mathsf{V}$$

If one repeats this for all the `b` bootstrap/simulated tables, then one gets `b` replicates of the matrices U^*, S^* and V^*, with which the confidence regions can be calculated, as explained in Section 2.3.

Bibliography

Adler, D., D. Murdoch and others (2017) *rgl: 3D Visualization Using OpenGL*. R package version 0.98.1.
https://CRAN.R-project.org/package=rgl.

Alvarez, R., M. Becue, J.J. Lanero and O. Valencia (2002) Results stability in textual analysis: Its application to the study of Spanish investiture speeches (1979-2000). *Proceedings of the Journees internationales d'Analyse statistique des Donnees Textuelles 2002*, 1–12.

Alvarez, R., M. Becue and O. Valencia (2004) Etude de la stabilite des valeurs propres de l'AFC d'un tableaux lexical au moyen de procedures de reechantillonage. *Proceedings of the Journees internationales d'Analyse statistique des Donnees Textuelles 2004*, 42–51.

Alvarez, R., M. Becue and O. Valencia (2006) Partial bootstrap in CA: correction of the coordinates. Application to textual data. *Proceedings of the Journees internationales d'Analyse statistique des Donnees Textuelles 2006*, 43–53.

Darroch, J.N. (1974) Multiplicative and additive interaction in contingency tables. *Biometrika* **61** (2), 207–214.

Escofier, B. (1984) Analyse factorielle en reference a un modele: Application a l'analyse de tableaux d'echanges. *Revue de Statistique Appliquee* **32** (4), 25–36.

Fisher, R.A. (1940) The precision of discriminant functions. *Annals of Eugenics* **10** (1), 422–429.

Gilula, Z. and S.J. Haberman (1988) The analysis of multivariate contingency tables by restricted canonical and restricted association models. *Journal of the American Statistical Association* **83** (403), 760–771.

Greenacre, M. (2017) *Correspondence analysis in practice, Third edition.* Boca Raton: Chapman and Hall/CRC.

Hirschfeld, H. O. (1935) A connection between correlation and contingency. *Proceedings of the Cambridge Philosophical Society* **31** (4), 520–524.

Kroonenberg, P.M. and C.J. Anderson (2006) Additive and multiplicative models for three-way contingency tables: Darroch (1974) revisited. In: M. Greenacre and J. Blasius (eds), *Multiple Correspondence Analysis and Related Methods.* Boca Raton: Chapman and Hall/CRC, 455–502.

Lebart, L. (2004) Validite des visualisations de donnees textuelles. *Proceedings of the Journees internationales d'Analyse statistique des Donnees Textuelles 2004*, 708–715.

Murdoch, D. and E.D. Chow (2013) *ellipse: Functions for drawing ellipses and ellipse-like confidence regions.* R package version 0.3-8.
https://CRAN.R-project.org/package=ellipse.

Soetaert, K. (2014) *diagram: Functions for visualising simple graphs (networks), plotting flow diagrams.* R package version 1.6.3.
https://CRAN.R-project.org/package=diagram.

Van der Heijden, P.G.M., A. De Falguerolles and J. De Leeuw (1989) A combined approach to contingency table analysis using correspondence analysis and log-linear analysis. *Applied Statistics* **38** (2), 249–292.

Venables, W.N. and B.D. Ripley (2002) *Modern applied statistics with S.* New York: Springer.