Principal Component Analysis (PCA)

Ph. Grosjean <phgrosjean@sciviews.org>

March 24, 2012

Part I
Introduction

Principal Component Analysis, or PCA is a widely used method to explore linear associations among variables of large datasets. There is, unfortunately, no consistent implementation of this technique in R, which is even more a problem because the numerous additional R packages that provide enhanced versions of PCA, or additional tools, have no consistent template to start with. In the **stats** package, there are two inconsistent implementations called `princomp()` and `prcomp()` that both create S3 objects of the same name. There are a few methods available, like `print()`, `summary()`, `plot()`, `predict()` or `biplot()`. The whole set is rather deceptive and produces less interesting plots than other (more specialized) software can do. For instance, there is nothing to plot the so-called "graph of the variables" in the French terminology and you have to program it yourself.

Of course, there are several specialized R packages available that provide more powerful and/or more extended implementations, among others: **ade4**, **FactoMineR** and **vegan**. Each of these packages has a totally different approach: **ade4** creates a `"c("pca", "dudi")"` S3 object and proposes nice graphs but has an interface that is completely inconsistent with usual R analyses (no optional formula interface, exotic names of arguments, non-standard handling of missing data, etc.). Object orientation and name of objects are obscure and do not facilitate first use of the PCA in **ade4**. A PCA is done, indeed, using the `dudi.pca()` function (or possibly, `nipals()`, but that creates a different `"nipals"` object). The same remarks can be made about the interface of functions in **FactoMineR**: they use strange arguments and do not respect the general organization of analyses in R (an object constructs the analysis, possibly defined using a formula; methods summarize or plot the results piece by piece). At least, name of function and object related to PCA are clear in **FactorMineR**: `PCA()`! There is also a non conventional handling of missing observations. But the function is powerful and loows for a lot of investigations around the PCA. In **vegan**, there is no PCA function, but a redundancy analysis `rda()`, which reduces to a classical PCA when arguments `X =` and `Y =` are missing. It creates a `"c("cca", "rda")"` S3 object which is not optimized at all for holding pure PCA data (many unnecessary items in it for a PCA). Finally, **labdsv** uses the default `prcomp()`, but it wraps it

into a "pca" S3 object, in order to define additional plotting methods that are consistent with the other analyses and objects in that package. Note that both "pca" S3 objects in **ade4** and **labdsv** are completely inconsistent, and you are likely to get very bad results in case you load both packages and mix their respective methods!

So, given that chaotic set of PCA functions in R, would it be possible to design an object with minimal code that reuses code in the **stats** package (princomp() and prcomp()), provides a couple of additional methods to make decent variables and individuals plots (possibly with ellipses or conver hulls for subgroups) in a way that a whole analysis would be easy to perform and to read in R code? We will try to do so in the present **SciViews** package.

First of all, we want to keep things simple. That is, we will design an S3 object, and not start from a complex S4 UML, as it is done for instance in the **rrcov** package. It would be nice to name this object "pca" and we should be able to make it compatible with both "princomp" and **labdsv**'s "pca" (but not with prcomp() that names loadings and scores components rotation and x, respectively. Also, that "pca" S3 object could **not** be compatible with **ade4**'s "pca" object. Moreover, neither **ade4**, nor **labdsv** use a namespace (as for versions available at 2010-02-06). Hopefully, **ade4** does not define methods specific for its "pca" object, except score.pca() for the score() generic function defined in the same package (and not elsewhere). Thus, we could define scores(), with 's', as for the corresponding item in **princomp** object without clash. Note that, if we don't use a nf item in our "pca" object, the **ade4**'s score() function inadvertently applied to our object fails with the error message: "Error in x$nf : $ operator is invalid for atomic vectors".

1 The SciViews' pcomp object

We finally choose "pcomp" as name of our object, but it inherits from "pca" and "princomp", because "pca" is already used in **ade4** and **labdsv** (with conflicting definitions), "PCA" is used in **FactoMineR** and "Pca" defines S4 objects in **rrcov** (and pCa is something totally different in **seacarb** package). Moreover, pcomp() is closer to prcomp() and princomp() as it is supposed to be a wrapper over these two (default) PCA functions in R.

The "pcomp" S3 object is a list with components:

- loadings: (also required for **labdsv**'s "pca" object). This is $rotation in "prcomp", and a "loadings" object in "princomp",
- scores: (also required for **labdsv**'s "pca" object). Note for scores in princomp, components are Comp.1, Comp.2, etc., in prcomp, it is PC1, PC2, ..., as well as in pca) => use PC1, PC2, ... This is $x in prcomp.For princomp(), the argument scores = TRUE (by default) must be used to get this!
- sdev: (also required for **labdsv**'s "pca" object). princomp() uses names (to rename into PC1, PC2, ...), while prcomp() does not,
- totdev: the total deviance, as required to be compliant with **labdsv**'s "pca" object.

- ▷ `n.obs`: the number of observations,
- ▷ `center`: (use 0 for all, if not centered),
- ▷ `scale`: (use 1 for all, if not scaled),
- ▷ `method`: currently only either ``` ``svd'' ``` (and the computation is the same as `prcomp()`), or ``` ``eigen'' ``` (and the computation is the same as `princomp()`),
- ▷ `call`: the matched call,
- ▷ `na.action`: if relevant.

This document needs to be finalized!