Practical course using the R software

Introduction to genetic data analysis in R

Thibaut Jombart

Abstract

This practical course is meant as a short introduction to genetic data analysis using R [10]. After describing the main types of data, we illustrate how to perform some basic population genetics analyses, and then go through constructing trees from genetic distances and performing standard multivariate analyses. The practical uses mostly the packages *adegenet* [6], *ape* [9] and *ade4* [1, 3, 2], but others like *genetics* [11] and *hierfstat* [5] are also required.

Contents

1 Let's start

1.1 Loading the packages

Before going further, we shall make sure that all we need is installed on the computer. Launch R, and make sure that the version being used is at least 2.12.0 by typing:

```
> R.version.string
```

```
[1] "R version 2.12.0 (2010-10-15)"
```

The next thing to do is check that relevant packages are installed. To load an installed package, use the `library` instruction; for instance:

```
> library(adegenet)
```

loads *adegenet* if it is installed (and issues an error otherwise). To get the version of a package, use:

```
> packageDescription("adegenet", fields = "Version")
```

```
[1] "1.2-9"
```

adegenet version should read 1.2-8 or greater.

In case a package would not be installed, you can install it using `install.packages`. To install all the required dependencies, specify `dep=TRUE`. For instance, the following instruction should install *adegenet* with all its dependencies (it can take up to a few minutes, so don't run it unless *adegenet* is not installed):

```
> install.packages("adegenet", dep = TRUE)
```

Using the previous instructions, load (and install if required) the packages *adegenet*, *ade4*, *ape*, *genetics*, and *hierfstat*.

1.2 How to get information?

There are several ways of getting information about R in general, or about *adegenet* in particular. The function `help.search` is used to look for help on a given topic. For instance:

```
> help.search("Hardy-Weinberg")
```

replies that there is a function `HWE.test.genind` in the *adegenet* package, other similar functions in *genetics* and *pegas*. To get help for a given function, use `?foo` where 'foo' is the function of interest. For instance (quotes can be removed):

```
> `?`(spca)
```

will open the manpage of the spatial principal component analysis [7]. At the end of a manpage, an 'example' section often shows how to use a function. This can be copied and pasted to the console, or directly executed from the console using `example`. For further questions concerning R, the function `RSiteSearch` is a powerful tool to make an online research using keywords in R's archives (mailing lists and manpages).

adegenet has a few extra documentation sources. Information can be found from the website (`http://adegenet.r-forge.r-project.org/`), in the 'documents' section, including two tutorials, a manual which includes all manpages of the package, and a dedicated mailing list with searchable archives. To open the website from R, use:

```
> adegenetWeb()
```

The same can be done for tutorials, using `adegenetTutorial` (see manpage to choose the tutorial to open).

You will also find a listing of the main functions of the package typing:

```
> `?`(adegenet)
```

Note that you can also browse help pages as html pages, using:

```
> help.start()
```

To go to the *adegenet* page, click 'packages', 'adegenet', and 'adegenet-package'.

Lastly, several mailing lists are available to find different kinds of information on R; to name a few:

R-help (`https://stat.ethz.ch/mailman/listinfo/r-help`): general questions about R

R-sig-genetics (`https://stat.ethz.ch/mailman/listinfo/r-sig-genetics`): genetics in R

adegenet forum (`https://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegenet-forum`): adegenet and multivariate analysis of genetic markers

2 Handling the data

2.1 Data formats

Two principal types of genetic data can be handled in R. The first one is (preferably aligned) *DNA sequences*, and the second one is *genetic markers*.

DNA sequences can be used to calibrate models of evolution and compute genetic distances, which can in turn be used for phylogenetic reconstruction or in multivariate analyses. In R, DNA sequences are best handled as `DNAbin` objects, in the *ape* package. See `?DNAbin` for more details about this class. After loading *ape*, we load the dataset `woodmouse`:

```
> library(ape)
> data(woodmouse)
> woodmouse

15 DNA sequences in binary format stored in a matrix.

All sequences of same length: 965

Labels: No305 No304 No306 No0906S No0908S No0909S ...

Base composition:
    a     c     g     t
0.307 0.261 0.126 0.306
```

Use `str`, `unclass` and `attributes` to explore the content of the `DNAbin` object. Convert the dataset to a matrix of characters using `as.character`. What is the size of this new object (use `object.size`)? Compare it to the original `DNAbin` object. Why is this class optimal for handling DNA sequences?

Genetic markers are features of DNA sequences that vary between individuals. There is not a single type of genetic marker: some are codominant (all alleles are visible), some are dominant (some alleles hide the presence of others); some have many alleles (like microsatellites), and others are most often binary (like SNPs). In general, codominant (more informative) markers are the rule, and dominant markers become more and more rare.

Both types of markers can be handled in R using the class `genind` implemented in *adegenet*. This class uses the S4 system of class definition and methods, with which R users are generally not familiar. Load the dataset nancycats:

```
> library(adegenet)
> data(nancycats)
> nancycats

   #####################
   ### Genind object ###
   #####################
- genotypes of individuals -
```

```
S4 class:  genind
@call: genind(tab = truenames(nancycats)$tab, pop = truenames(nancycats)$pop)

@tab:  237 x 108 matrix of genotypes

@ind.names: vector of  237 individual names
@loc.names: vector of  9 locus names
@loc.nall: number of alleles per locus
@loc.fac: locus factor for the  108 columns of @tab
@all.names: list of  9 components yielding allele names for each locus
@ploidy:  2
@type:  codom

Optionnal contents:
@pop:  factor giving the population of each individual
@pop.names:  factor giving the population of each individual

@other: a list containing: xy
```

Usually, the slots of S4 objects can be accessed by '@', which replaces the $ operator used in S3 classes (e.g. in lists). For genind objects, both can be used. Documentation of the class can be accessed by typing ?'genind-class'. Using names and @ (or $), browse the content of nancycats; what information is contained in this object?

A more simple dataset can be used to understand how information is coded. Here, we create a simple dataset with two diploid individuals (in rows), and two markers (columns).

```
> dat <- data.frame(locus1 = c("A/T", "T/T"), locus2 = c("G/G",
+     "C/G"))
> dat

  locus1 locus2
1    A/T    G/G
2    T/T    C/G


> x <- df2genind(dat, sep = "/")
> x


   #####################
   ### Genind object ###
   #####################
- genotypes of individuals -

S4 class:  genind
@call: df2genind(X = dat, sep = "/")

@tab:  2 x 4 matrix of genotypes

@ind.names: vector of  2 individual names
@loc.names: vector of  2 locus names
@loc.nall: number of alleles per locus
@loc.fac: locus factor for the  4 columns of @tab
@all.names: list of  2 components yielding allele names for each locus
@ploidy:  2
@type:  codom

Optionnal contents:
@pop:  - empty -
```

```
@pop.names:  - empty -

@other: - empty -
```

Look at the @tab component of the genind object. To extract this table with original labels, use truenames. How is information stored? Does the Euclidean distance between the rows of this matrix make some biological sense? This is a canonical way of storing information for any dominant markers, irrespective of the degree of ploidy.

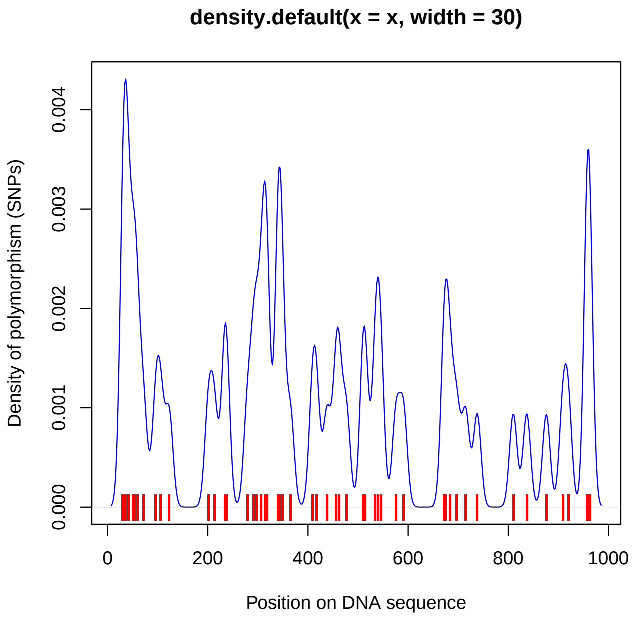
It sometimes happens that analyses are performed at a population level, rather than at an individual level. In such cases, `genpop` objects are used. These objects contain allele numbers per populations rather than relative frequencies, but are otherwise similar to `genind` objects. `genpop` can be obtained from `genind` objects using genind2genpop. Use this function to compute allele counts for the cat colonies of Nancy (data `nancycats`), and compare the the structure of the obtained object to the `genind` object.

2.2 Importing / exporting data

DNA sequences with usual formats are read using read.dna from the *ape* package.

Genetic markers come in a wider variety of formats. import2genind can be used to import some of the most frequent, and less twisted formats. However, it is also possible to convert data from a data.frame to a genind using df2genind. Using the example above, create a data.frame with two tetraploid (i.e. 4 alleles) individuals and two loci. Note that alleles can be given any name, and do not need to be letters. Then, used df2genind to convert it to a genind object. Use truenames and genind2df to check that the data conversion has worked.

It also happens that genetic markers are derived from DNA sequences. In such a case, only sites that vary between individuals are retained; these are called Single Nucleotide Polymorphism (SNPs). They can be extracted from DNAbin objects using DNAbin2genind. Use this function to extract the polymorphic sites from the woodmouse dataset. The locus names correspond to the position of the polymorphic site. Use this information to plot the distribution of polymorphism along the DNA sequence; you should arrive at something along the lines of (using density with a width of 30):



What can we say about the distribution of SNPs on the DNA sequence?

2.3 Manipulating data

It is often useful to be able to manipulate genetic marker data in different ways. There is no particular manipulation for DNAbin objects, which are essentially matrices or lists. genind objects benefit from different facilities for manipulating data. First, the [operator can be used to subset individuals and alleles; this is done like for a matrix, and is based on the @tab slot. For instance, to retain only the first 10 individuals of the nancycats dataset, use:

```
> nancycats[1:10, ]

   #####################
   ### Genind object ###
   #####################
- genotypes of individuals -

S4 class:  genind
@call: .local(x = x, i = i, j = j, drop = drop)

@tab:  10 x 108 matrix of genotypes

@ind.names: vector of  10 individual names
@loc.names: vector of  9 locus names
@loc.nall: number of alleles per locus
@loc.fac: locus factor for the  108 columns of @tab
@all.names: list of  9 components yielding allele names for each locus
@ploidy:  2
@type:  codom
```

```
Optionnal contents:
@pop:  factor giving the population of each individual
@pop.names:  factor giving the population of each individual

@other: a list containing: xy
```

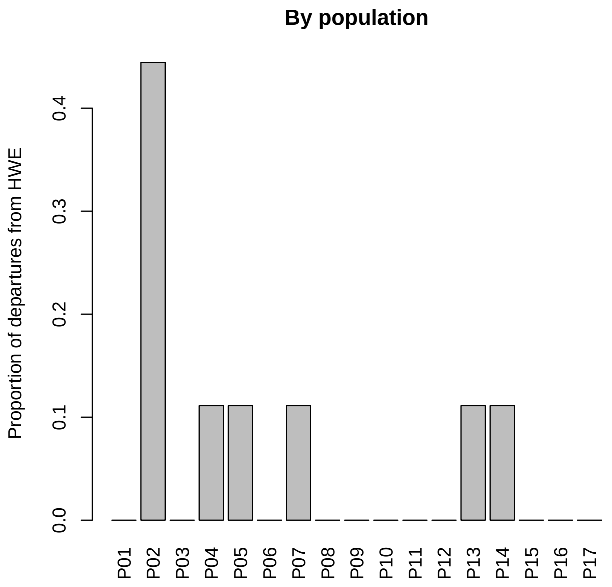
Data can be split by population using `seppop`, and by locus using `seploc`. Indication of the population is obtained or set using `pop`. `repool` can be used to gather the genotypes stored in different `genind` objects into a single `genind`. Reading the documentation, interprete the following command line:

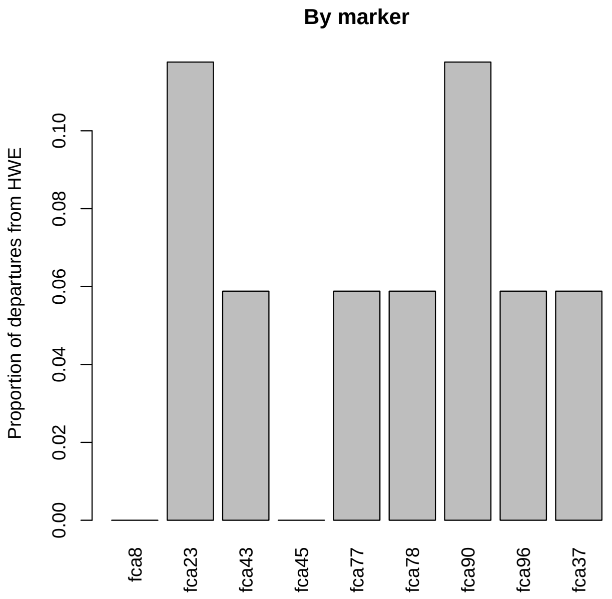
```
> temp <- seppop(nancycats)
> x <- repool(lapply(temp, function(e) e[sample(1:nrow(e$tab),
+     5, replace = FALSE)]))
```

What does the produced object contain? How can this be useful when analysing genetic data?

3 Basic population genetics

This section aims at introducing some basic population genetics tools. The first one consists in testing random mating within groups by testing Hardy-Weinberg equilibirum (HWE). Using `HWE.test.genind`, test HWE in the cat colonies of the dataset `nancycats`. Since this corresponds to a lot of tests (1 per group and locus), as a first approximation, ask for returning only the p-values. Compute the number of significant departures from HWE (p.value < 0.001) per locus, and then per population (use `apply`), and represent graphically the results. You should arrive at (in less than 3 command lines):





What are your conclusions? What can we say about the hypothesis of random mating within these cat colonies?

To investigate the data further, we can measure and test the significance of the genetic differentiation of the cat colonies. Use fstat to compute the F_{st} of these populations. This is a measure of population differentiation, which can be roughly interpreted as the proportion of genetic variance explained by differences between groups. Values less than 0.05 are often considered as negligible.

Then, test the significance of the overall group differentiation using gstat.randtest, and plot the results; are there differences between groups?

4 Making trees

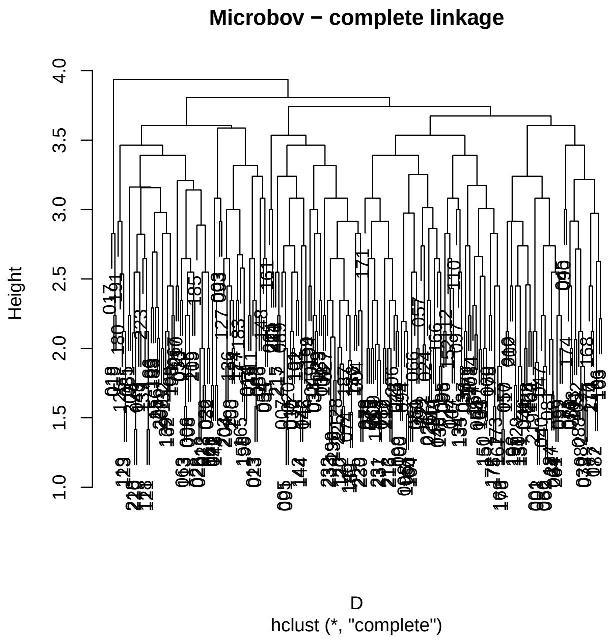
Genetic data are often represented using trees. In this section, we won't tackle phylogenetic reconstruction based on maximum likelihood or parsimony, which should be covered in a practical of its own. For this, see the well-documented package *phangorn*. In the following, we illustrate briefly how to contruct tree representations of genetic distances.

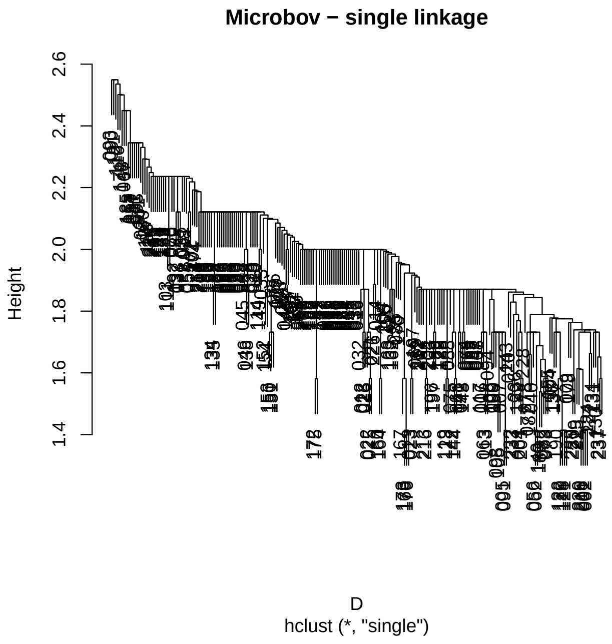
4.1 Hierarchical clustering

Several algorithm of hierarchical clustering are implemented in the function hclust. This function requires a distance matrix produced by the function dist, or similar function (e.g. dist.dna, dist.genpop).

We will try to describe the genetic variability in the cat colonies of Nancy using clustering methods. First, we need to compute a distance matrix. Replace missing data in nancycats using na.replace, and then compute the Euclidean distances between individuals using dist on the table of relative allele frequencies ($tab of the object in which NAs have been replaced). Use hclust to compute a clustering with complete linkage, and one with single linkage. How do they compare? How would you interprete these graphs? Can you assess the number of actual genetic clusters in these data?

As a comparison, repeat these analyses with the dataset microbov, which contains genotypes of various cattle breeds in France and Africa.





Species in the `microbov` dataset are well differentiated. How good is hierarchical clustering at displaying genetic structures?

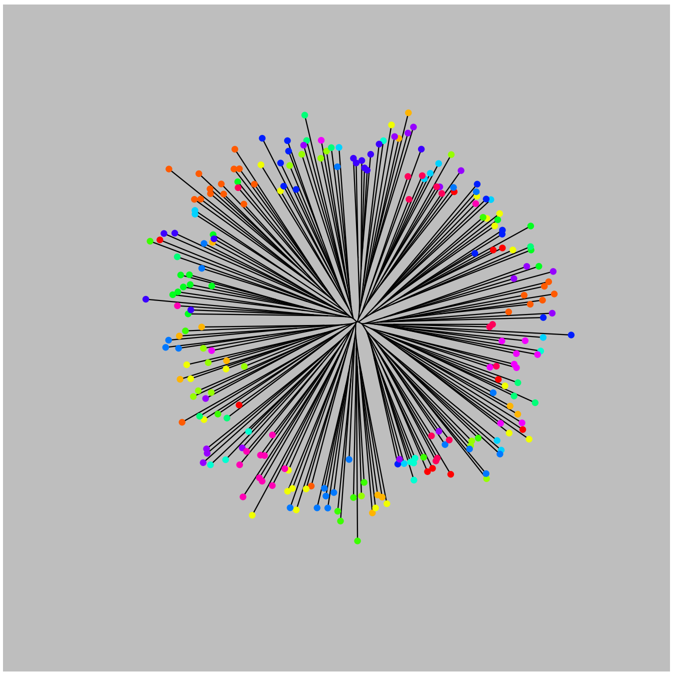
4.2 ape and phylogenies

ape is the main package for phylogenetic analyses. It implements various methods for reconstructing trees based on genetic distances. It also provides nice graphical functions for plotting trees.

Use the genetic distances obtained for the nancycat data to compute a Neighbour-Joining tree (nj). Then, plot the tree, specifying that you want to plot an unrooted tree, and use tiplabels to represent colonies by coloured dots. To define one color for each colony, use:

```
> myCol <- rainbow(17)[as.integer(pop(nancycats))]
```

Since some colours are very light, you are best using a grey background for the plot. The final result should ressemble:

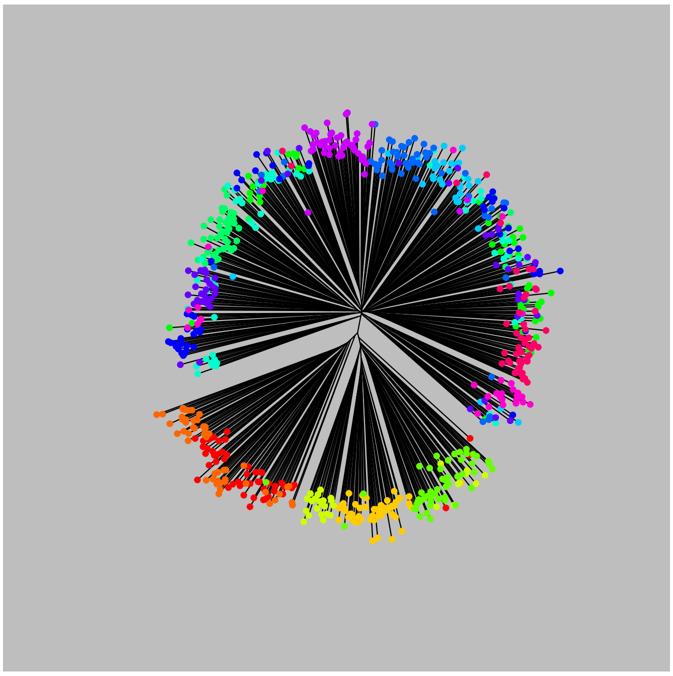


For a comparison, repeat these analyses with the cattle breed data (microbov).

To define one color for each breed of the dataset, use:

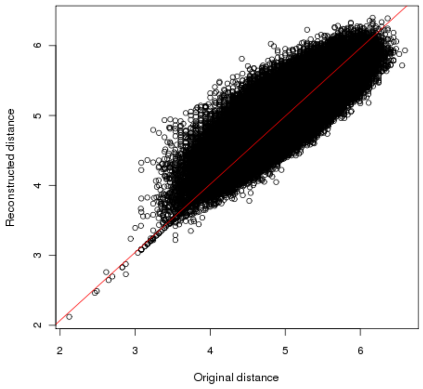
```
> myCol <- rainbow(15)[as.integer(pop(microbov))]
```

You should obtain a tree like this one:



There are two species, *Bos taurus* and *Bos indicus*, in these cattle breeds; the species information is stored in `microbov$other$spe`. Use the same approach to represent the species information on the tips of the tree. Is this classification reflected by a clear-cut structure on the tree?

The quality of a tree is unfortunately rarely questioned in terms of how well the real distances are represented by the tree. The literature is even full of examples of analyses based on distances computed from trees, while the original distances that had generated the tree could have been used. A simple way of assessing how good a tree is at representing a set of distances is simply comparing the original distances to the reconstructed distances (sum of branch lengths between tips). The latter can be computed using `cophenetic`. Use this function to plot both distances (`microbov` data), after converting the distances into vectors using `as.vector`; you should obtain the following figure:



The red line indicates the regression of the original distance onto the reconstructed (function `lm`). Perform this regression, and use `summary` to compute various statistics for this model, including the R^2. What percentage of the variance of the original distances is represented by the tree reconstruction? Is this satisfying? Shall we always rely on a tree representation of a distance matrix?

5 Multivariate Analysis

5.1 ade4

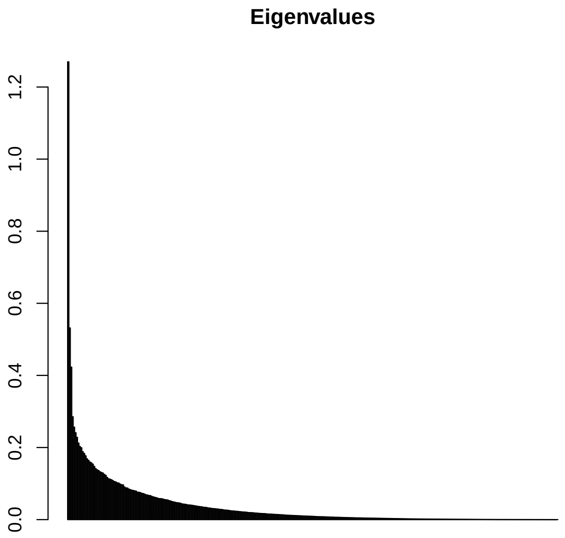
There are a few of packages implementing multivariate analyses in R. One of the most complete is *ade4*, with which *adegenet* is interfaced. It has been developped around the "duality diagram" representation [4, 2], which emphasizes that most multivariate analyses can be unified under a single general algorithm. Alternatively, some statisticians refer to this algorithm as "generalized PCA".

In *ade4*, multivariate analyses all use the same underlying algorithm (`as.dudi`) and produce an object with the class `dudi` (for "duality diagram"). Methods such as Principal Component Analysis (PCA) or Principal Coordinates Analysis (PCoA) are implemented by functions where the methods' abbreviation is preceeded by `dudi`; for instance, PCA is implemented by `dudi.pca`, while PCoA is implemented by `dudi.pco`.

Apart from a few specific methods like spatial PCA (`spca`, [7]) or Discriminant Analysis of Principal Components (`dapc`, [8]), multivariate methods are not implemented in *adegenet*. Procedures from *ade4* are used after transforming `genind` or `genpop` objects in an appropriate way, which often consists in replacing missing data, and computing and standardizing allele frequencies before analysis.

5.2 Principal Component Analysis (PCA)

PCA can be used on standardized allele frequencies, at the individual or populational level. The function scaleGen is used to standardize the data and replace possible missing data. Use it on the microbov dataset to obtain a table of centred, but not scaled allele frequencies. Then, perform the PCA of these data using dudi.pca. Be careful to set the argument scale to FALSE to avoid erasing the scaling choice made in scaleGen.



The function dudi.pca displays a barplot of eigenvalues and asks for a number of retained principal components. In general, eigenvalues represent the amount of genetic diversity — as measured by the multivariate method being used — represented by each principal component (PC). Here, each eigenvalue is the variance of the corresponding PC.

Once you have selected the number of retained axes, your PCA should ressemble:

```
Duality diagramm
class: pca dudi
$call: dudi.pca(df = X, center = FALSE, scale = FALSE, scannf = FALSE,
    nf = 3)

$nf: 3 axis-components saved
$rank: 341
eigen values: 1.27 0.5317 0.423 0.2853 0.2565 ...
  vector length mode    content
1 $cw    373    numeric column weights
2 $lw    704    numeric row weights
```

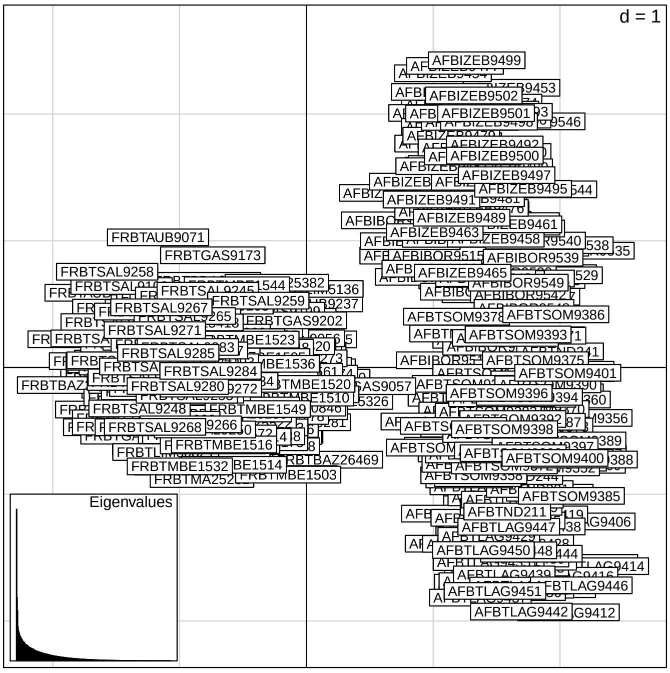
```
3 $eig   341    numeric eigen values

  data.frame nrow ncol content
1 $tab       704  373  modified array
2 $li        704  3    row coordinates
3 $l1        704  3    row normed scores
4 $co        373  3    column coordinates
5 $c1        373  3    column normed scores
other elements: cent norm
```

The printing of dudi object is not always explicit. The essential items for the analyses we will use are:

- ⋆ $eig: eigenvalues of the analysis
- ⋆ $c1: principal axes, containing the loadings of the variables (alleles)
- ⋆ $li: principal components, containing the scores of the individuals

The function scatter provides a simultaneous representation of individuals and variables. However, it does not provide very informative graphics when there are too many items to be plotted. To represent individuals only, use s.label, and use the argument clab to change the size of the labels; for instance:

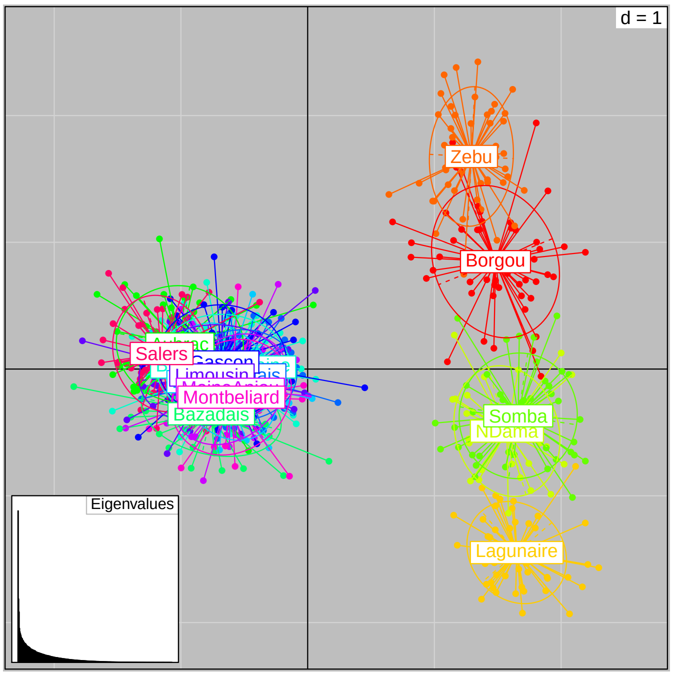
```
> s.label(pcaX$li, clab = 0.75)
> add.scatter.eig(pcaX$eig, 3, 1, 2)
```



This is better, but far from perfect yet. In this case, it would be better to represent results by breed. This can be achieved using s.class. In addition to the coordinates (the same as in s.label), this function needs a factor grouping the observations. This can be obtained easily using pop on the object microbov. To be even fancier, you can define one color per group using rainbow (for other palettes, see ?rainbow):

```
> myCol <- rainbow(length(levels(pop(microbov))))
```

and then pass it as the color argument in s.class; the result should ressemble this:



Represent the results for the axes 1-2, and then 2-3. How can you interprete these principal components? What is the major factor of genetic differentiation in these cattle breeds? What is the second one? What is the third one?

In PCA, the eigenvalues indicate the variance of the corresponding principal component. Verify that this is indeed the case, for the first and second principal components. Note that this is also, up to a constant, the mean squared Euclidean distance between individuals. This is because (for $x \in \mathbb{R}^n$):

$$var(x) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j)^2}{2n(n-1)}$$

This again can be checked; assuming this analyis is called pcaX:

```
> pcaX$eig[1]

[1] 1.269978
```

```
> pc1 <- pcaX$li[, 1]
> n <- length(pc1)
> 0.5 * mean(dist(pc1)^2) * ((n - 1)/n)
```

```
[1] 1.269978
```

We sometimes want to express eigenvalues as a percentage of the total variation in the data. What is the percentage of variance explained by the first three eigenvalues?

Allele contributions can sometimes be informative. The basic graphics for representing allele loadings is s.arrow. Use it to represent the results of the PCA; is this informative? An alternative is offered by loadingplot, which represents one axis at a time. Use it to represent the squared loadings of the analysis; how does it compare to previous representation? The function returns invisibly some information; adapting the arguments and saving the returned value, identify the 2% alleles contributing most to showing the diversity within African breeds. You should find:

```
[1] "ETH152.197"  "INRA32.178"  "HEL13.182"   "MM12.119"    "CSRM60.093"
[6] "TGLA227.079" "TGLA126.119" "TGLA122.150"
```

5.3 Principal Coordinates Analysis (PCoA)

We have just seen that the variance is, up to a constant, equal to the mean squared distance between all pairs of observations. Therefore, maximizing the variance along an axis should be exactly the same as maximizing the sum of all squared Euclidean distances. This latter maximization is the objective of Principal Coordinates Analysis (PCoA), also known as Metric Multidimensional Scaling (MDS). This method is implemented in *ade4* by dudi.pco. After scaling the relative allele frequencies of the microbov dataset, perform this analysis.

Represent the principal components ($li) of this analysis and compare them to the PCs of the previous PCA. What is the correlation between them? Compare the scatter plots of both analyses; are there any differences? In practice, in which cases should we prefer PCA over PCoA, or the contrary?

As an exercise, compute genetic similarities based on the proportion of shared alleles (propShared) on the microbov data. Transform the similarities into a distance (no dedicated function for this), and perform a PCoA of the resulting distance. Is this distance Euclidean? If not, it needs to be. Transform it into an Euclidean distance (using cailliez) and redo the PCoA. How do the results compare to the previous analyses?

References

[1] D. Chessel, A-B. Dufour, and J. Thioulouse. The ade4 package-I- one-table methods. *R News*, 4:5–10, 2004.

[2] S. Dray and A.-B. Dufour. The ade4 package: implementing the duality diagram for ecologists. *Journal of Statistical Software*, 22(4):1–20, 2007.

[3] S. Dray, A.-B. Dufour, and D. Chessel. The ade4 package - II: Two-table and K-table methods. *R News*, 7:47–54, 2007.

[4] Y. Escoufier. The duality diagramm : a means of better practical applications. In P. Legendre and L. Legendre, editors, *Development in numerical ecology*, pages 139–156. NATO advanced Institute , Serie G .Springer Verlag, Berlin, 1987.

[5] J. Goudet. HIERFSTAT, a package for R to compute and test hierarchical F-statistics. *Molecular Ecology Notes*, 5:184–186, 2005.

[6] T. Jombart. adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics*, 24:1403–1405, 2008.

[7] T. Jombart, S. Devillard, A.-B. Dufour, and D. Pontier. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, 101:92–103, 2008.

[8] Thibaut Jombart, Sebastien Devillard, and Francois Balloux. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. *BMC Genetics*, 11(1):94, 2010.

[9] E. Paradis, J. Claude, and K. Strimmer. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004.

[10] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[11] G. R. Warnes. The genetics package. *R News*, 3(1):9–13, 2003.