Introduction to genetic data analysis using

Thibaut Jombart*

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

August 17, 2016

Abstract

This practical introduces basic multivariate analysis of genetic data using the *adegenet* and `ade4` packages for the R software. We briefly show how genetic marker data can be read into R and how they are stored in *adegenet*, and then introduce basic population genetics analysis and multivariate analyses. These topics are covered in further depth in the *basics* tutorial, which can be accessed from the *adegenet* website or by typing `adegenetTutorial("basics")` in `R`.

*tjombart@imperial.ac.uk

Contents

1 Getting started **3**
1.1 Installing the package . . . 3
1.2 Getting help . . . 3

2 Importing data **5**

3 First look at the data **7**

4 Basic population genetics analyses **14**
4.1 Testing for Hardy-Weinberg equilibrium . . . 14
4.2 Assessing population structure . . . 15

5 Multivariate analyses **21**
5.1 Principal Component Analysis (PCA) . . . 21
5.2 Principal Coordinates Analysis (PCoA) . . . 29

6 To go further **31**

1 Getting started

1.1 Installing the package

Before going further, we shall make sure that *adegenet* is installed and up to date. The current version of the package is 2.0.1. Make sure you have a recent version of R ($\geq$ 3.2.1) by typing:

```
R.version.string

## [1] "R version 3.3.1 (2016-06-21)"
```

Then, to install the stable version of *adegenet* with dependencies, type:

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

If *adegenet* was already installed, you can ensure that it is up-to-date using:

```
update.packages(ask=FALSE)
```

As an alternative, you can install the current devel version of *adegenet*, which incorporates the latest changes and improvements. To do so, you first need the package `devtools` installed:

```
install.packages("devtools")
```

and then type:

```
library(devtools)
install_github("thibautjombart/adegenet")
```

We can now load the useful packages using:

```
library("adegenet")
library("ape")
library("pegas")
```

1.2 Getting help

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

```
help.search("Monmonier")
```

replies that there is a handful of functions implementing Monmonier's algorithm (for detecting spatial genetic boundaries) in the *adegenet* package. To get help for a given function, use `?foo` where `foo` is the function of interest. For instance:

```
?monmonier
```

will open up the help of the main function implementing the algorithm. 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, and sometimes directly executed from the console using `example` (for examples with a short runtime). For further questions concerning R, the function `RSiteSearch` is a powerful tool for making online researches 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 several tutorials and a manual which compiles 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 `?adegenetTutorial` for how to choose the tutorial to open). Similarly, bug reports or feature requests can be made using Github's issue system, accessible via:

```
adegenetIssues()
```

You will also find an overview of the main functionalities 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:

- *adegenet forum*: adegenet and genetic data analysis in R.
  https://lists.r-forge.r-project.org/cgi-bin/mailman/listinfo/adegenet-forum

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

- *R-sig-genetics*: population genetics in R.
  https://stat.ethz.ch/mailman/listinfo/r-sig-genetics

- *R-sig-phylo*: phylogenetics in R.
  https://stat.ethz.ch/mailman/listinfo/r-sig-phylo

2 Importing data

Data can be imported from a wide range of formats, including those of popular population genetics software (GENETIX, STRUCTURE, Fstat, Genepop), or from simple dataframes of genotypes. Polymorphic sites can be extracted from both nucleotide and amino-acid sequences, with special methods for handling genome-wide SNPs data with miminum RAM requirements. Data can be stored using two main classes of object:

- `genind`: allelic data for individuals stored as (integer) allele counts

- `genpop`: allelic data for groups of individuals ("populations") stored as (integer) allele counts

Typically, data are first imported to form a `genind` object, and potentially aggregated later into a `genpop` object. Given any grouping of individuals, one can convert a `genind` object into a `genpop` using `genind2genpop`.

The main functions for obtaining a `genind` object are:

- `import2genind`: GENETIX/Fstat/Genepop files → `genind` object

- `read.structure`: STRUCTURE files → `genind` object

- `df2genind`: `data.frame` of alleles → `genind` object

- `DNAbin2genind`: `DNAbin` object → `genind` object (conserves SNPs only)

- `alignment2genind`: `alignment` object → `genind` object (conserves SNPs/polymorphic amino-acid sites only)

Here, we will use a dataset distributed with *adegenet*, which can be loaded using:

```
data(nancycats)
cats <- nancycats
cats
```

```
## /// GENIND OBJECT /////////
##
##  // 237 individuals; 9 loci; 108 alleles; size: 145.3 Kb
##
##  // Basic content
##    @tab:  237 x 108 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 8-18)
##    @loc.fac: locus factor for the 108 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: genind(tab = truenames(nancycats)$tab, pop = truenames(nancycats)$pop)
##
##  // Optional content
##    @pop: population of each individual (group size range: 9-23)
##    @other: a list containing: xy
```

This `genind` object contains microsatellite genotypes of 237 cats from various colonies in Nancy, France (see `?nancycats` for details).

3 First look at the data

`cats` is a `genind` object storing microsatellite data. You can compare its content to its the original dataset in GENETIX format, which you can visualize using:

```
file.show(system.file("files/nancycats.gtx",package="adegenet"))
```

`genind` objects store various information, including individual genotypes, labels for individuals, loci, and alleles, the ploidy of each individual, and some optional content such as population membership, spatial coordinates, etc. The content of `genind` objects can be accessed, and in some cases changed, using simple functions called "*accessors*":

- `nInd`: returns the number of individuals in the object; only for `genind`.
- `nLoc`: returns the number of loci.
- `nAll`: returns the number of alleles for each locus.
- `nPop`: returns the number of populations.
- `tab`: returns a table of allele numbers, or frequencies (if requested), with optional replacement of missing values; replaces the former accessor '`truenames`'.
- `indNames`†: returns/sets labels for individuals; only for `genind`.
- `locNames`†: returns/sets labels for loci.
- `alleles`†: returns/sets alleles.
- `ploidy`†: returns/sets ploidy of the individuals; when setting values, a single value can be provided, in which case constant ploidy is assumed.
- `pop`†: returns/sets a factor grouping individuals; only for `genind`.
- `strata`†: returns/sets data defining strata of individuals; only for `genind`.
- `hier`†: returns/sets hierarchical groups of individuals; only for `genind`.
- `other`†: returns/sets misc information stored as a list.

where † indicates that a replacement method is available using `<-`; for instance:

```
head(indNames(cats),10)

##  [1] "N215" "N216" "N217" "N218" "N219" "N220" "N221" "N222" "N223" "N224"

indNames(cats) <- paste("cat", 1:nInd(cats),sep=".")
head(indNames(cats),10)

##  [1] "cat.1"  "cat.2"  "cat.3"  "cat.4"  "cat.5"  "cat.6"  "cat.7"
##  [8] "cat.8"  "cat.9"  "cat.10"
```

The `cats` contains various information:

```
cats

## /// GENIND OBJECT /////////
##
##  // 237 individuals; 9 loci; 108 alleles; size: 145.3 Kb
##
##  // Basic content
##     @tab:  237 x 108 matrix of allele counts
##     @loc.n.all: number of alleles per locus (range: 8-18)
##     @loc.fac: locus factor for the 108 columns of @tab
##     @all.names: list of allele names for each locus
##     @ploidy: ploidy of each individual  (range: 2-2)
##     @type:  codom
##     @call: genind(tab = truenames(nancycats)$tab, pop = truenames(nancycats)$pop)
##
##  // Optional content
##     @pop: population of each individual (group size range: 9-23)
##     @other: a list containing: xy
```

Data are stored as allele counts in a matrix where rows are individuals and columns, alleles:

```
dim(cats@tab)

## [1] 237 108

class(cats@tab)

## [1] "matrix"

cats@tab[1:5,1:20]

##       fca8.117 fca8.119 fca8.121 fca8.123 fca8.127 fca8.129 fca8.131
## cat.1       NA       NA       NA       NA       NA       NA       NA
## cat.2       NA       NA       NA       NA       NA       NA       NA
## cat.3        0        0        0        0        0        0        0
## cat.4        0        0        0        0        0        0        0
## cat.5        0        0        0        0        0        0        0
##       fca8.133 fca8.135 fca8.137 fca8.139 fca8.141 fca8.143 fca8.145
## cat.1       NA       NA       NA       NA       NA       NA       NA
## cat.2       NA       NA       NA       NA       NA       NA       NA
## cat.3        0        1        0        0        0        1        0
## cat.4        1        1        0        0        0        0        0
## cat.5        1        1        0        0        0        0        0
```

```
##       fca8.147 fca8.149 fca23.128 fca23.130 fca23.132 fca23.136
## cat.1       NA       NA         0         0         0         1
## cat.2       NA       NA         0         0         0         0
## cat.3        0        0         0         0         0         1
## cat.4        0        0         0         0         0         0
## cat.5        0        0         0         0         0         0
```

Some accessors such as `locNames` may have specific options; for instance:

```
locNames(cats)

## [1] "fca8"  "fca23" "fca43" "fca45" "fca77" "fca78" "fca90" "fca96" "fca37"
```

returns the names of the loci, while:

```
temp <- locNames(cats, withAlleles=TRUE)
head(temp, 10)

##  [1] "fca8.117" "fca8.119" "fca8.121" "fca8.123" "fca8.127" "fca8.129"
##  [7] "fca8.131" "fca8.133" "fca8.135" "fca8.137"
```

returns the names of the alleles in the form 'loci.allele'.

The slot 'pop' can be retrieved and set using `pop`:

```
obj <- cats[sample(1:50,10)]
pop(obj)

##  [1] P01 P01 P04 P02 P03 P03 P03 P01 P02 P02
## Levels: P01 P02 P03 P04

pop(obj) <- rep("newPop",10)
pop(obj)

##  [1] newPop newPop newPop newPop newPop newPop newPop newPop newPop newPop
## Levels: newPop
```

Accessors make things easier. For instance, when setting new names for loci, the columns of `@tab` are renamed automatically:

```
head(colnames(tab(obj)),20)

##  [1] "fca8.117"  "fca8.119"  "fca8.121"  "fca8.123"  "fca8.127"
##  [6] "fca8.129"  "fca8.131"  "fca8.133"  "fca8.135"  "fca8.137"
## [11] "fca8.139"  "fca8.141"  "fca8.143"  "fca8.145"  "fca8.147"
## [16] "fca8.149"  "fca23.128" "fca23.130" "fca23.132" "fca23.136"
```

```
locNames(obj)

## [1] "fca8"  "fca23" "fca43" "fca45" "fca77" "fca78" "fca90" "fca96" "fca37"

locNames(obj)[1] <- "newLocusName"
locNames(obj)

## [1] "newLocusName" "fca23"        "fca43"        "fca45"
## [5] "fca77"        "fca78"        "fca90"        "fca96"
## [9] "fca37"

head(colnames(tab(obj)),20)

##  [1] "newLocusName.117" "newLocusName.119" "newLocusName.121"
##  [4] "newLocusName.123" "newLocusName.127" "newLocusName.129"
##  [7] "newLocusName.131" "newLocusName.133" "newLocusName.135"
## [10] "newLocusName.137" "newLocusName.139" "newLocusName.141"
## [13] "newLocusName.143" "newLocusName.145" "newLocusName.147"
## [16] "newLocusName.149" "fca23.128"        "fca23.130"
## [19] "fca23.132"        "fca23.136"
```

An additional advantage of using accessors is they are most of the time safer to use. For instance, `pop<-` will check the length of the new group membership vector against the data, and complain if there is a mismatch. It also converts the provided replacement to a factor, while the command:

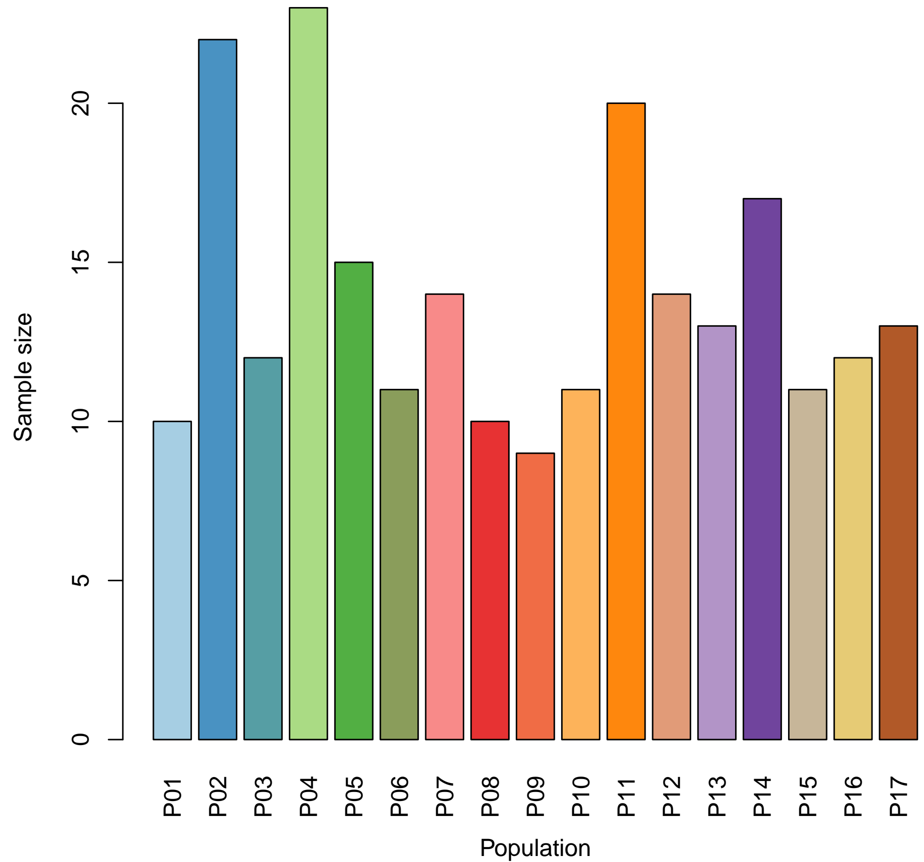
```
obj@pop <- rep("newPop",10)

## Error in (function (cl, name, valueClass) :  assignment of an object of
class "character" is not valid for @'pop' in an object of class "genind";
is(value, "factorOrNULL") is not TRUE
```

generates an error (since replacement is not a factor).

It is very easy, for instance, to obtain the sample sizes per populations using `table`:

```
head(pop(cats), 50)

##  [1] P01 P01 P01 P01 P01 P01 P01 P01 P01 P01 P02 P02 P02 P02 P02 P02 P02
## [18] P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P02 P03 P03
## [35] P03 P03 P03 P03 P03 P03 P03 P03 P03 P03 P04 P04 P04 P04 P04 P04
## 17 Levels: P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 ... P17

table(pop(cats))
```

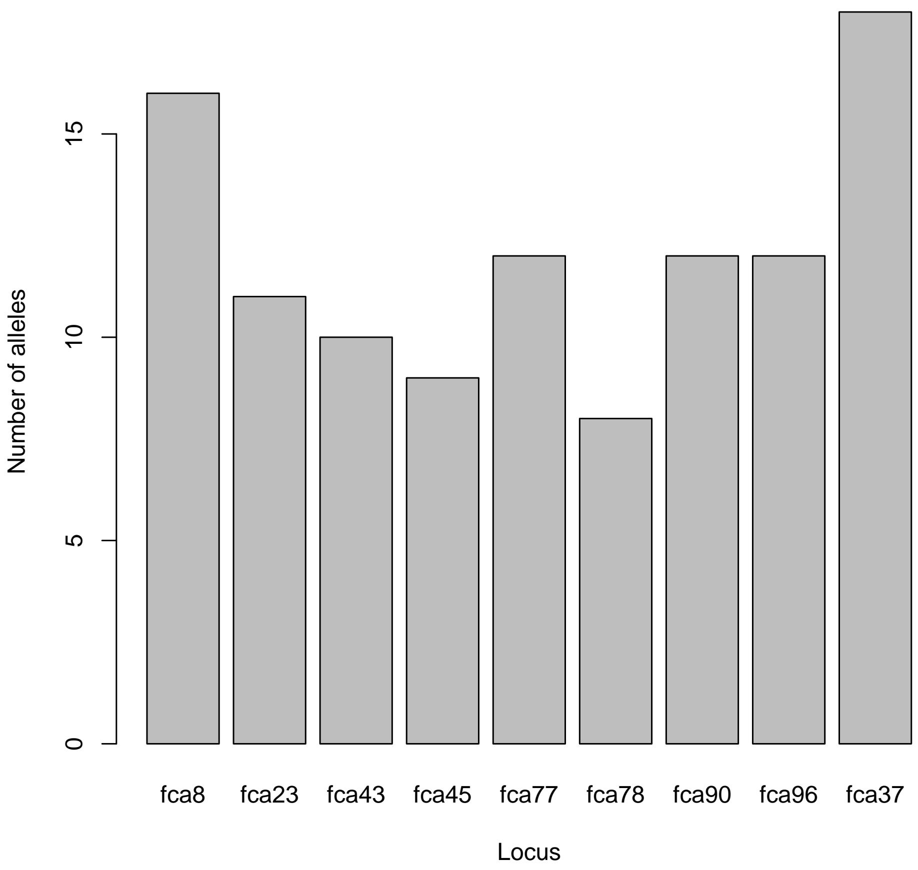
```
##
## P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17
##  10  22  12  23  15  11  14  10   9  11  20  14  13  17  11  12  13

barplot(table(pop(cats)), col=funky(17), las=3,
        xlab="Population", ylab="Sample size")
```

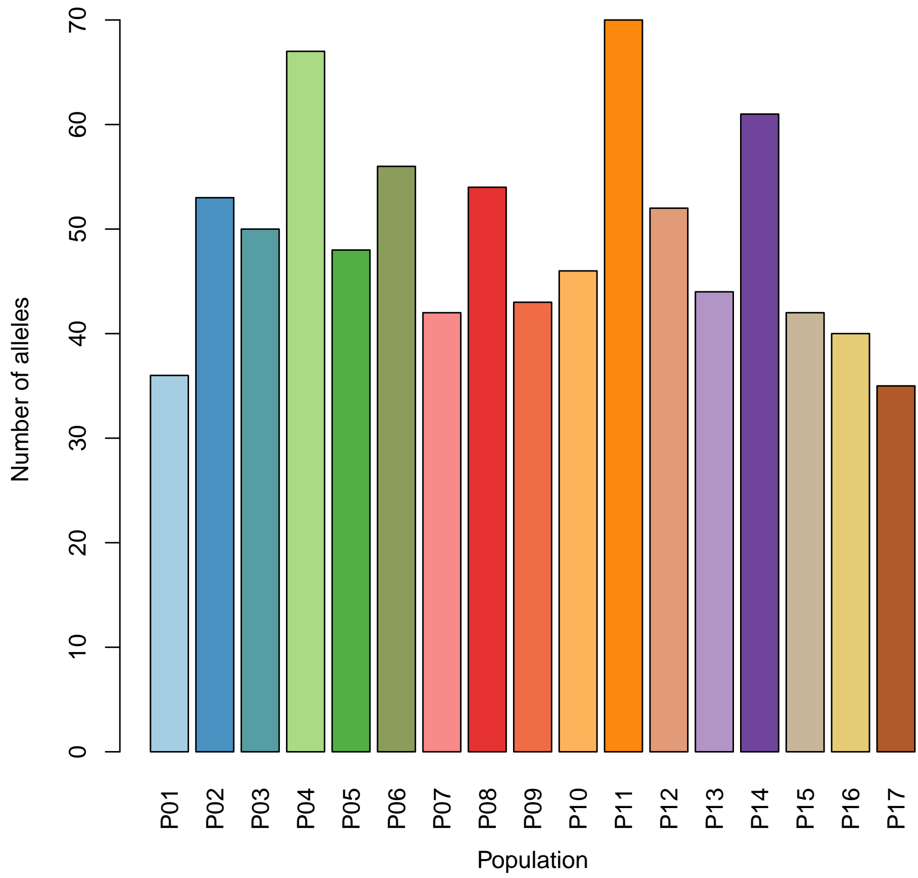


More information is available from the `summary`:

```
temp <- summary(cats)
```

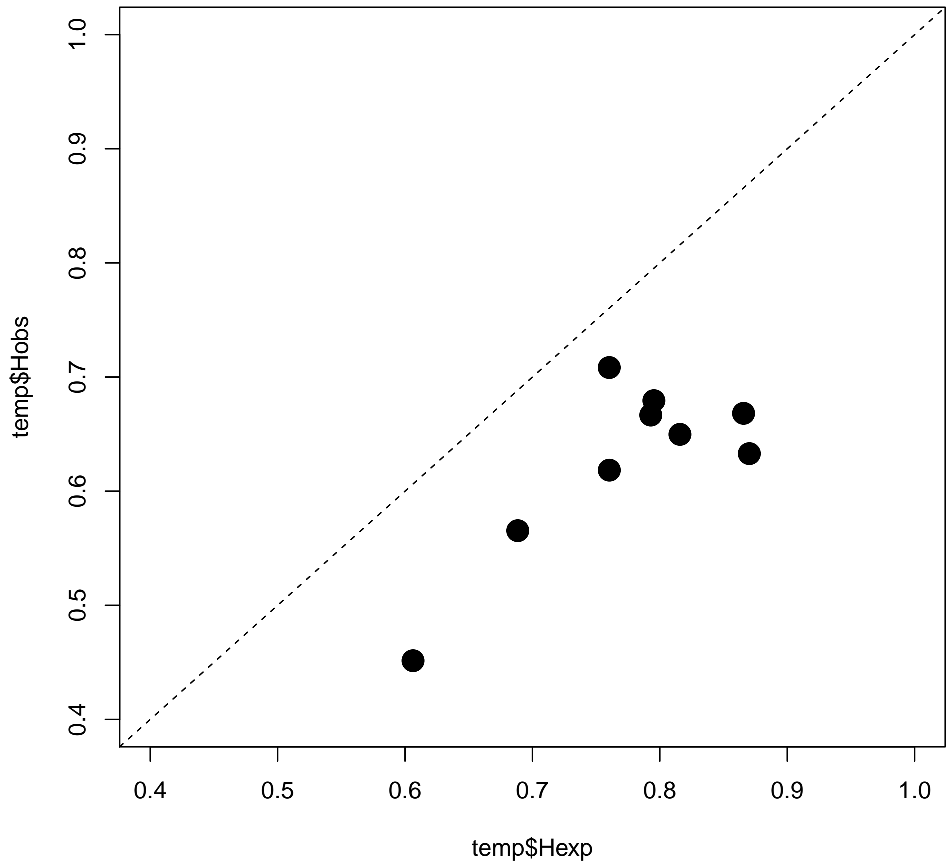
`temp` contains the information returned by `summary`. Using the same function as above, try displaying the number of alleles i) per locus and ii) per population. You should obtain something along the lines of:


Number of alleles
0
5
10
15
fca8
fca23
fca43
fca45
fca77
fca78
fca90
fca96
fca37
Locus



Knowing that `Hexp` and `Hobs` refer to the expected and observed heterozygosity, interpret:

```
plot(temp$Hexp, temp$Hobs, pch=20, cex=3, xlim=c(.4,1), ylim=c(.4,1))
abline(0,1,lty=2)
```



What can you say about the heterozygosity in these data? Is a statistical test needed?

4 Basic population genetics analyses

Deficit in heterozygosity can be indicative of population structure. In the following, we try to assess this possibility using classical population genetics tools.

4.1 Testing for Hardy-Weinberg equilibrium

Hardy-Weinberg equilibrium (HWE) defines, for a given locus, the expected frequencies of genotypes given the existing allele frequencies in a panmictic population. It relies on a number of strong assumptions about the studied population, including random mating, and the absence of selection, migration, and mutation.

The Hardy-Weinberg equilibrium (HWE) test is implemented for `genind` objects by `hw.test` in the package *pegas*. It provides two versions (parametric and non-parametric) of the test. Use both on the `nancycats` data. What is your conclusion concerning HWE in these data?

4.2 Assessing population structure

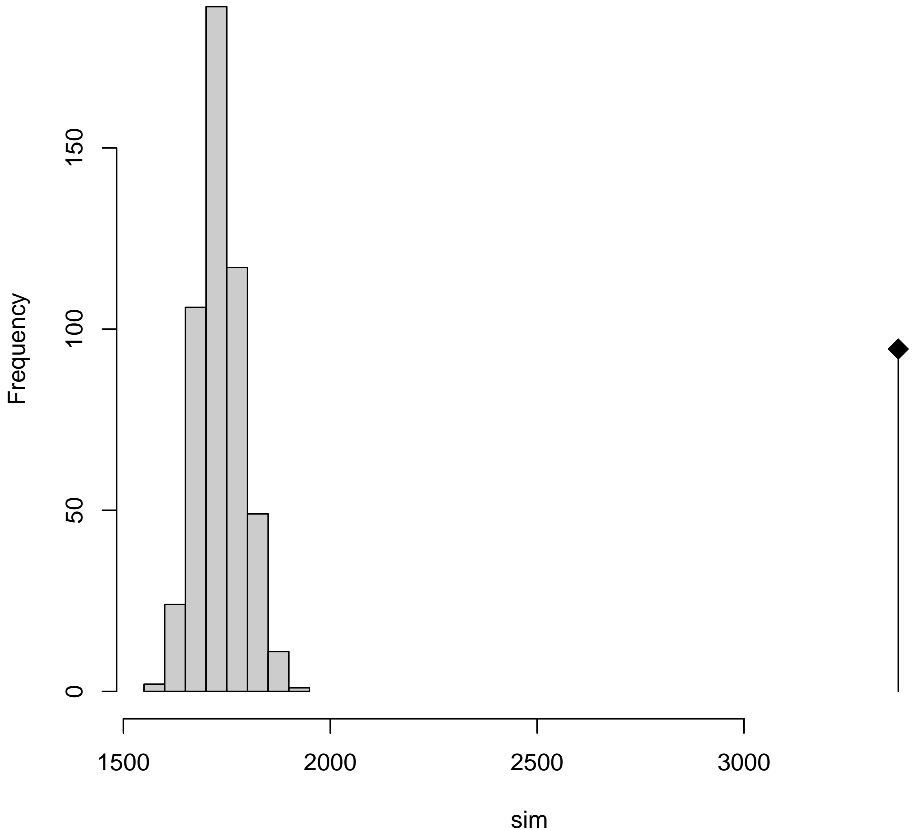
Population structure is traditionally measured and tested using F statistics, in particular the F_{st}, which measures population differentiation (as the proportion of allelic variance occuring between groups). The package *hierfstat* implements a wealth of F statistics and related tests, now designed to work natively with `genind` objects. The devel version of the package is required for these features. Install and load it using:

```
library(devtools)
install_github("jgx65/hierfstat")
```

```
library("hierfstat")
```

We can now use different methods for assessing population structure. We first compute overall F statistics, and then use Goudet's G statistics to test the existence of population structure. Try to interpret the following statistics and graphics:

```
fstat(cats)

##              pop       Ind
## Total 0.08494959 0.1952946
## pop   0.00000000 0.1205890

fstat(cats, fstonly=TRUE)

## [1] 0.08494959

cats.gtest <- gstat.randtest(cats)
cats.gtest

## Monte-Carlo test
## Call: gstat.randtest(x = cats)
##
## Observation: 3372.926
##
## Based on 499 replicates
## Simulated p-value: 0.002
## Alternative hypothesis: greater
##
##     Std.Obs Expectation    Variance
##    30.15915  1734.07191  2952.85547

plot(cats.gtest)
```



Is there some significant population structure? What is the proportion of the total genetic variance explained by the groups?

A more detailed picture can be seeked by looking at F_{st} values between pairs of populations. This can be done using the function `pairwise.fst`, which computes Nei's estimator of pairwise F_{st} defined as:

$$F_{st}(A,B) = \frac{H_t - (n_A H_s(A) + n_B H_s(B))/(n_A + n_B)}{H_t}$$

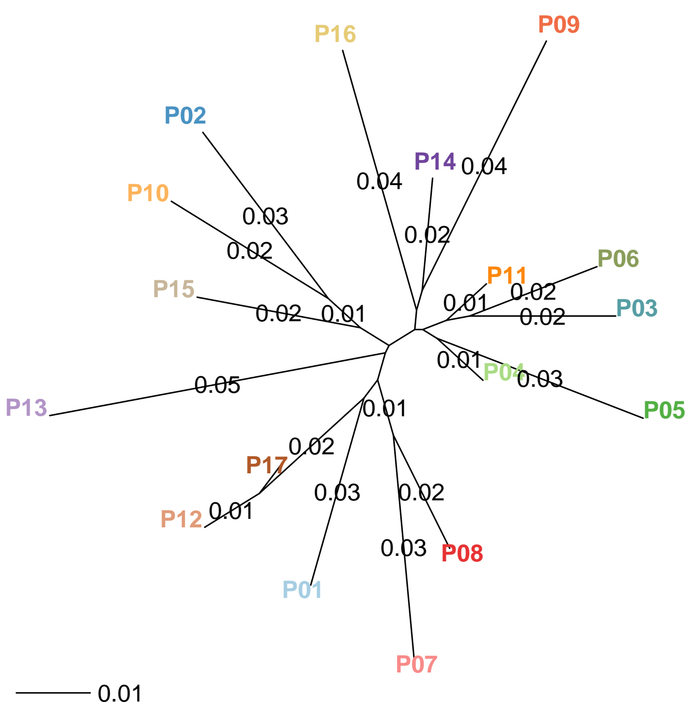
where A and B refer to the two populations of sample size n_A and n_B and respective expected heterozygosity $H_s(A)$ and $H_s(B)$, and H_t is the expected heterozygosity in the whole dataset. For a given locus, expected heterozygosity is computed as $1 - \sum p_i^2$, where p_i is the frequency of the ith allele, and the $\sum$ represents summation over all alleles. For multilocus data, the heterozygosity is simply averaged over all loci. Let us use this approach for the `cats` data:

```
cats.matFst <- pairwise.fst(cats,res.type="matrix")
cats.matFst[1:4,1:4]

##              P01          P02          P03          P04
```

```
## P01 0.00000000 0.08018500 0.07140847 0.04992548
## P02 0.08018500 0.00000000 0.08200880 0.06985472
## P03 0.07140847 0.08200880 0.00000000 0.02571561
## P04 0.04992548 0.06985472 0.02571561 0.00000000
```

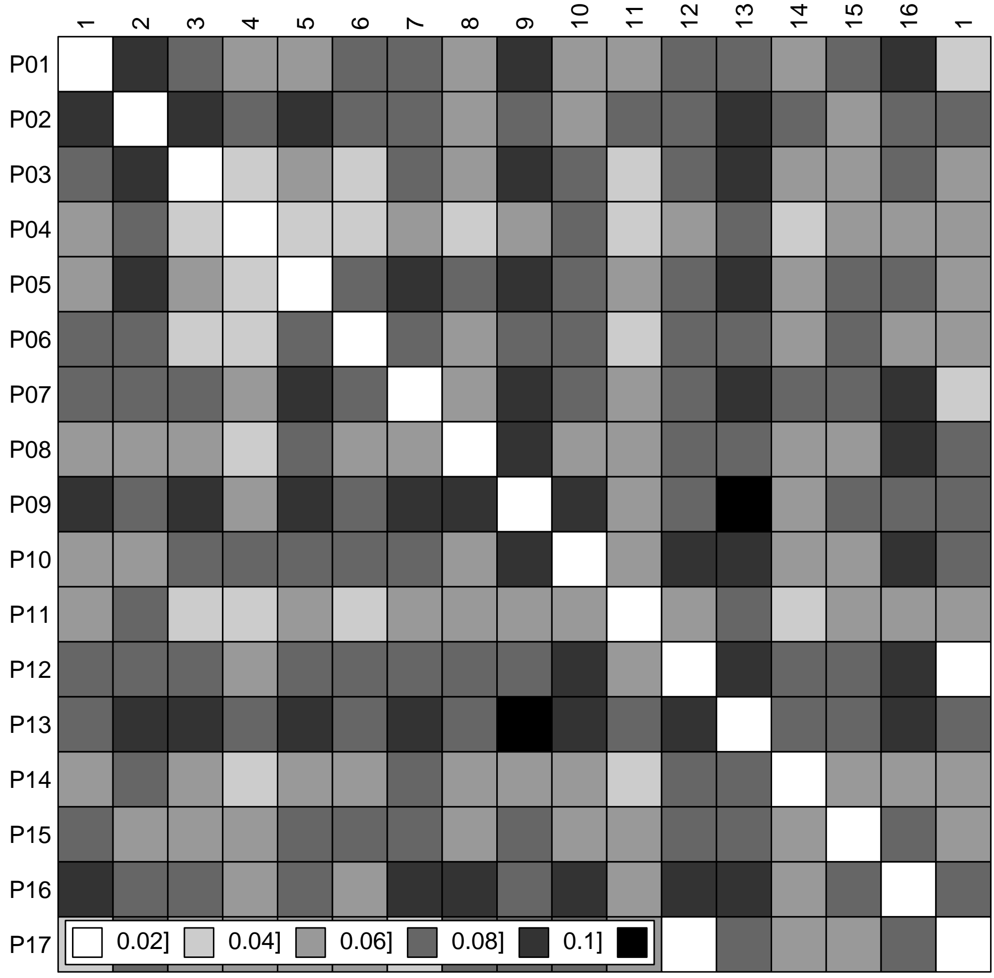
These values can be used as a measure of genetic distance between populations, which can in turn be used to build a tree. We use *ape* to do so:

```
cats.tree <- nj(cats.matFst)
plot(cats.tree, type="unr", tip.col=funky(nPop(cats)), font=2)
annot <- round(cats.tree$edge.length,2)
edgelabels(annot[annot>0], which(annot>0), frame="n")
add.scale.bar()
```



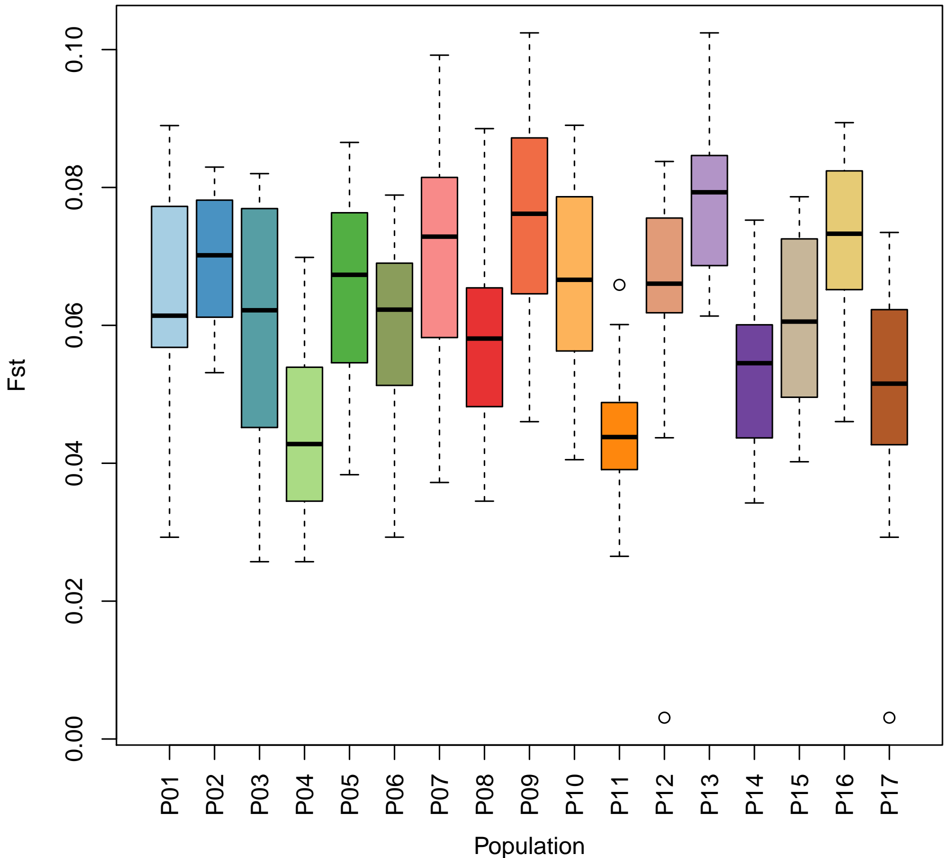
What can you say about the population structure? Is there an outlying group? To confirm your intuition, visualize the raw data using:

```r
table.paint(cats.matFst, col.labels=1:16)
```

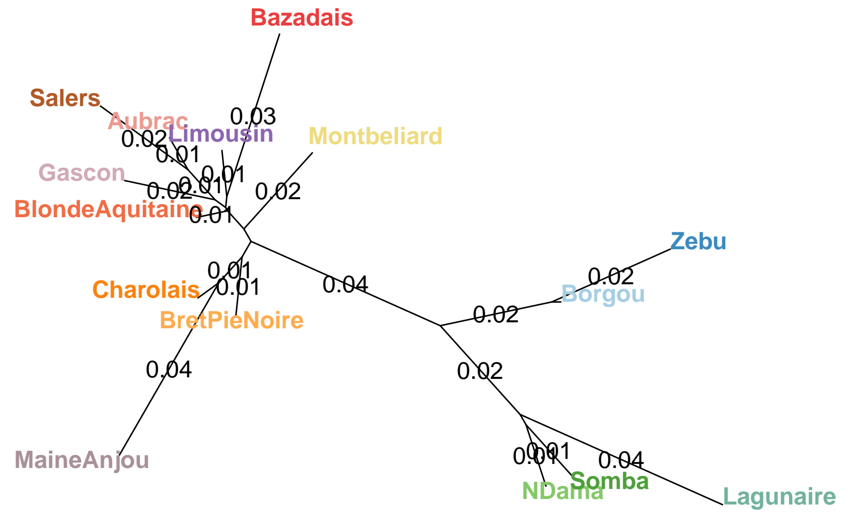


Interpret the following this figure:

```r
temp <- cats.matFst
diag(temp) <- NA
boxplot(temp, col=funky(nPop(cats)), las=3,
        xlab="Population", ylab="Fst")
```



As an exercise, try reproducing the same analysis using the dataset `microbov`, which contains genotypes of 704 cows from 15 breeds for 30 microsatellites loci (see `?microbov`). You should obtain something along the lines of:



0.02

What do you conclude?

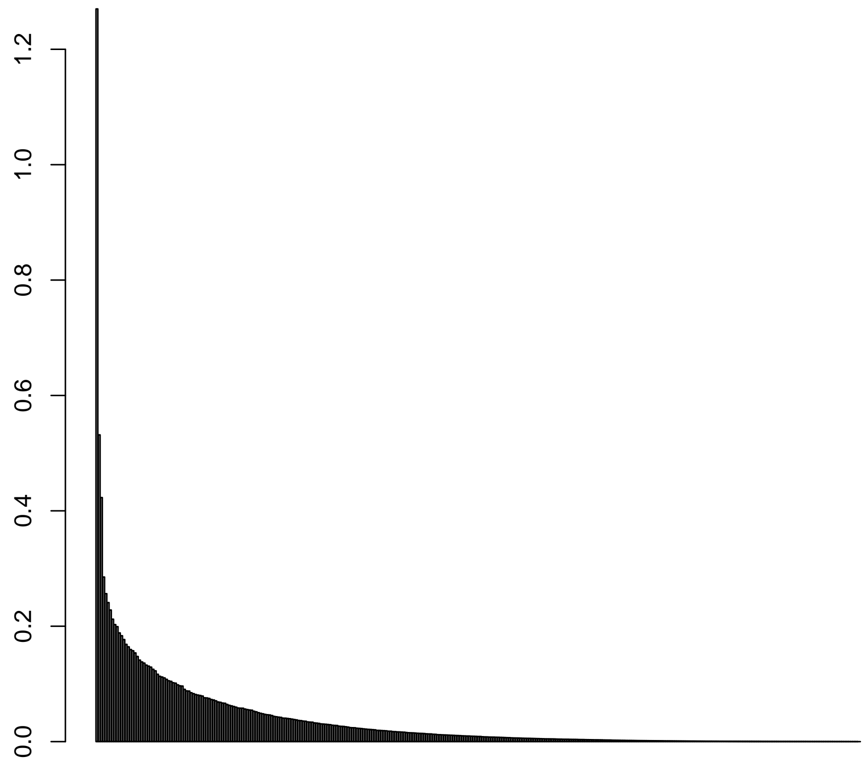
5 Multivariate analyses

5.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is the amongst the most common multivariate analyses used in genetics. Running a PCA on `genind` object is straightforward. One needs to first extract allelic data (as frequencies) and replace missing values using the accessor `tab` and then use the PCA procedure (`dudi.pca`). Let us use this approach on the `microbov` data. Let us first load the data:

```
data(microbov)
x.cows <- tab(microbov, freq=TRUE, NA.method="mean")
```

```
pca.cows <- dudi.pca(x.cows, center=TRUE, scale=FALSE)
```



The function `dudi.pca` displays a barplot of eigenvalues (the *screeplot*) 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. A sharp decrease in the eigenvalues is usually indicative of the boundaries between relevant structures and random noise. Here, how many axes would you retain?

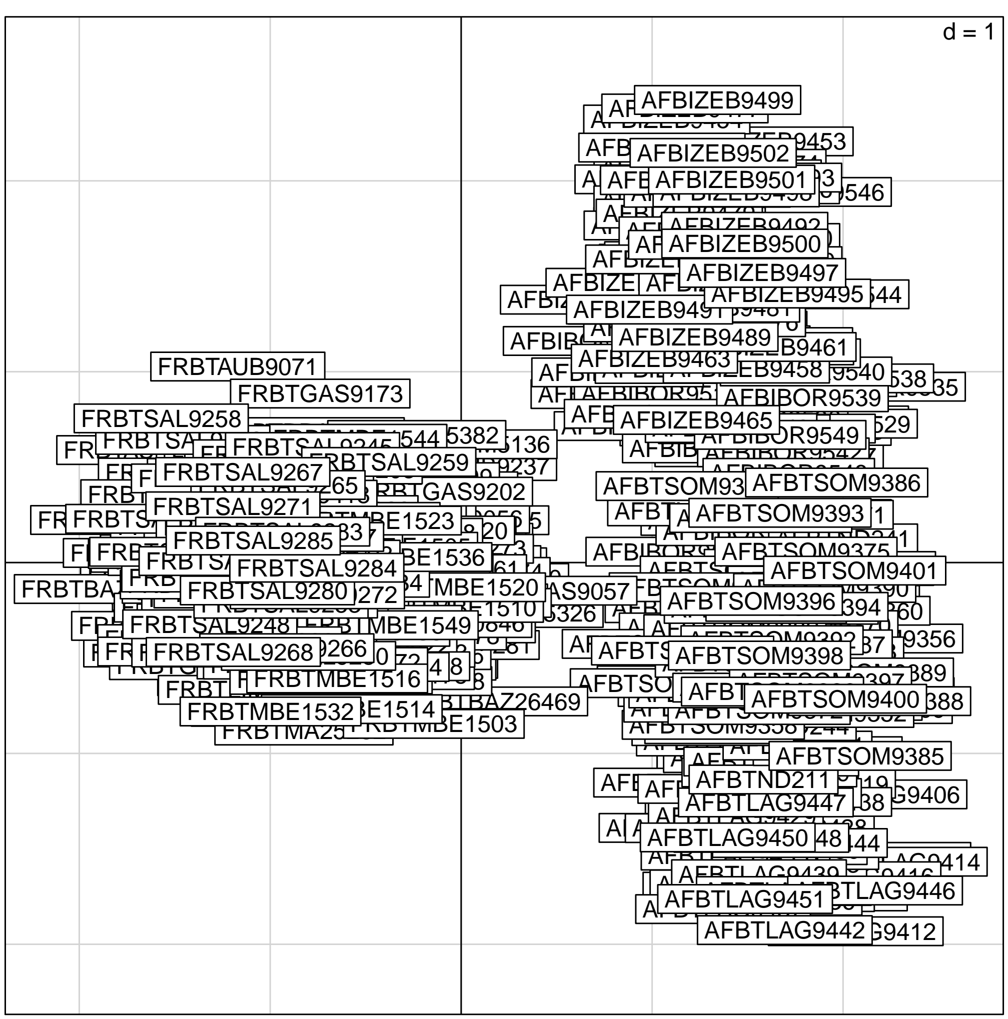
```
pca.cows

## Duality diagramm
## class: pca dudi
## $call: dudi.pca(df = x.cows, center = TRUE, 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 output object `pca.cows` is a list containing various information; of particular interest are:

- `$eig`: the eigenvalues of the analysis, indicating the amount of variance represented by each principal component (PC).
- `$li`: the principal components of the analysis; these are the synthetic variables summarizing the genetic diversity, usually visualized using scatterplots.
- `$c1`: the allele loadings, used to compute linear combinations forming the PCs; squared, they represent the contribution to each PCs.

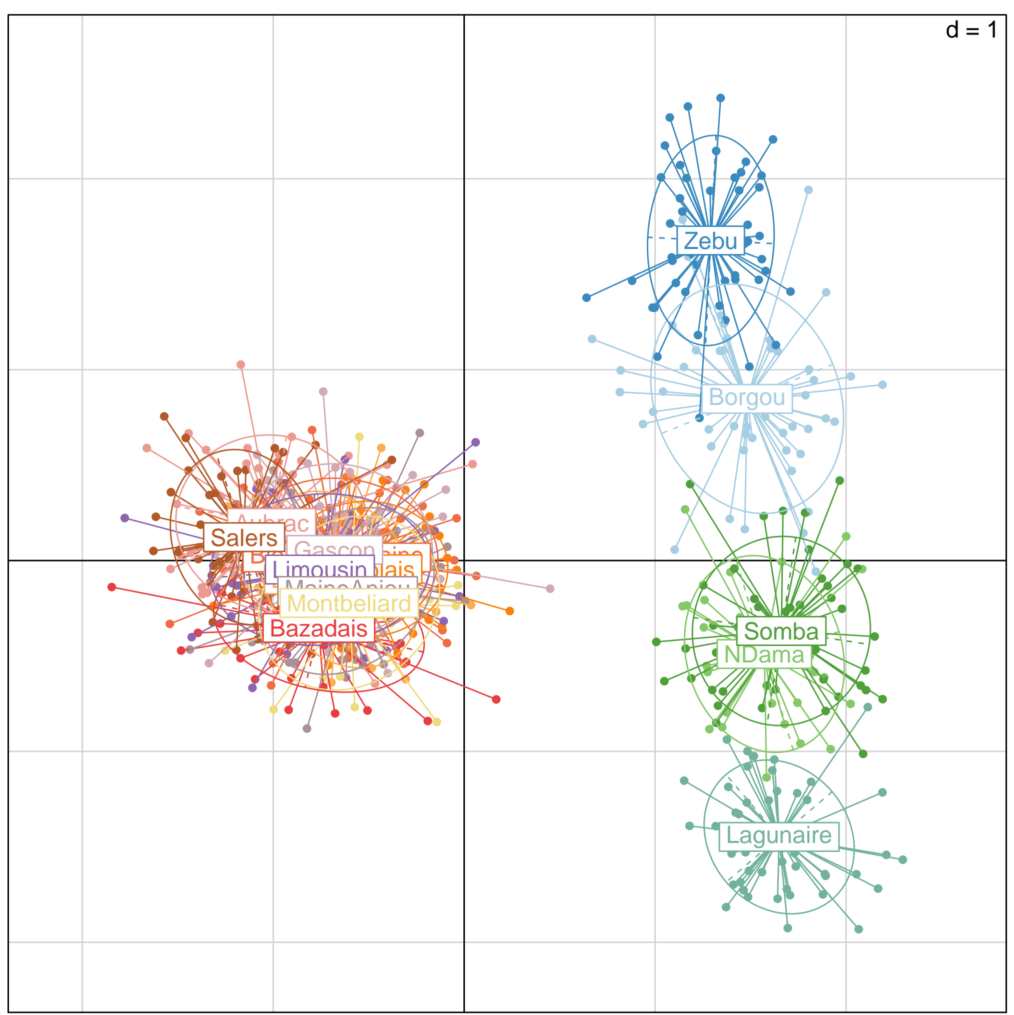
Coordinates of individual genotypes onto the principal axes can be visualized using `s.label`:

```
s.label(pca.cows$li)
```



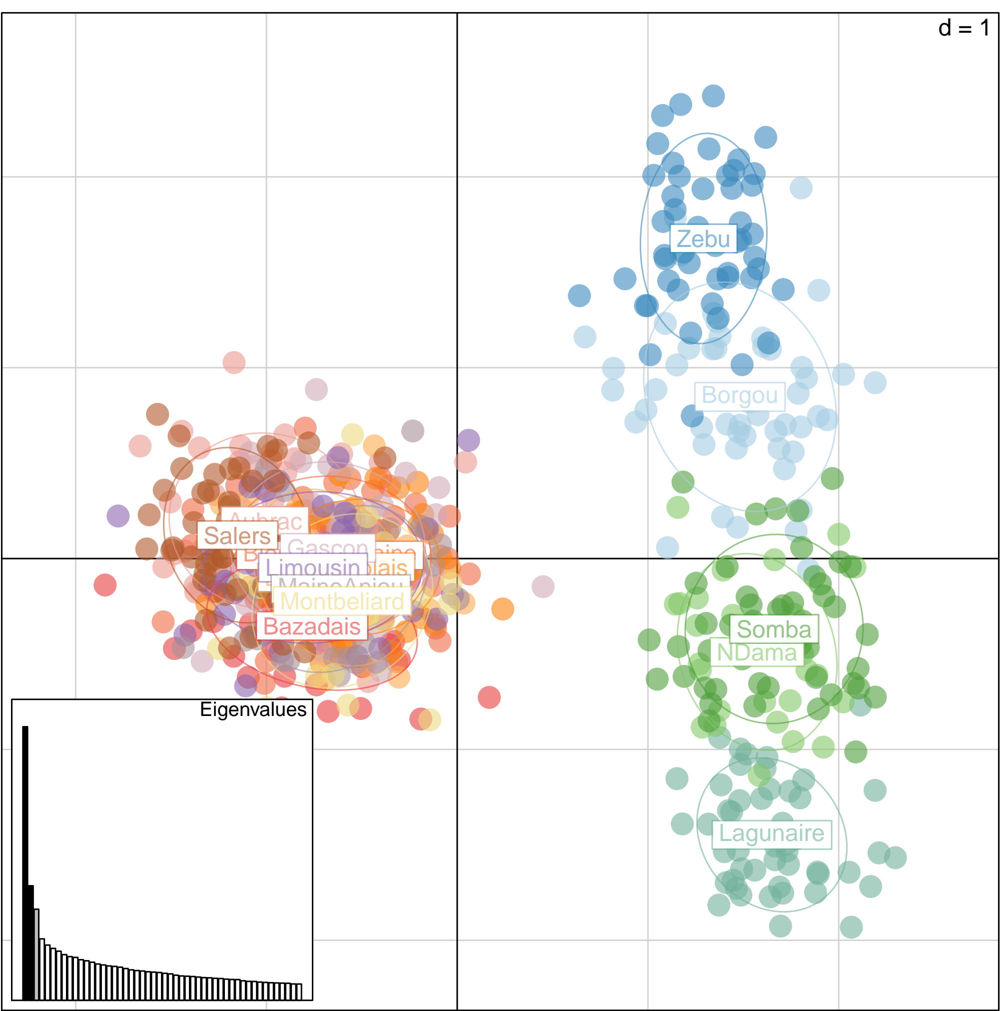
This is, however, not very usefull here. Let us add group information using `s.class`:

```
s.class(pca.cows$li, fac=pop(microbov), col=funky(15))
```



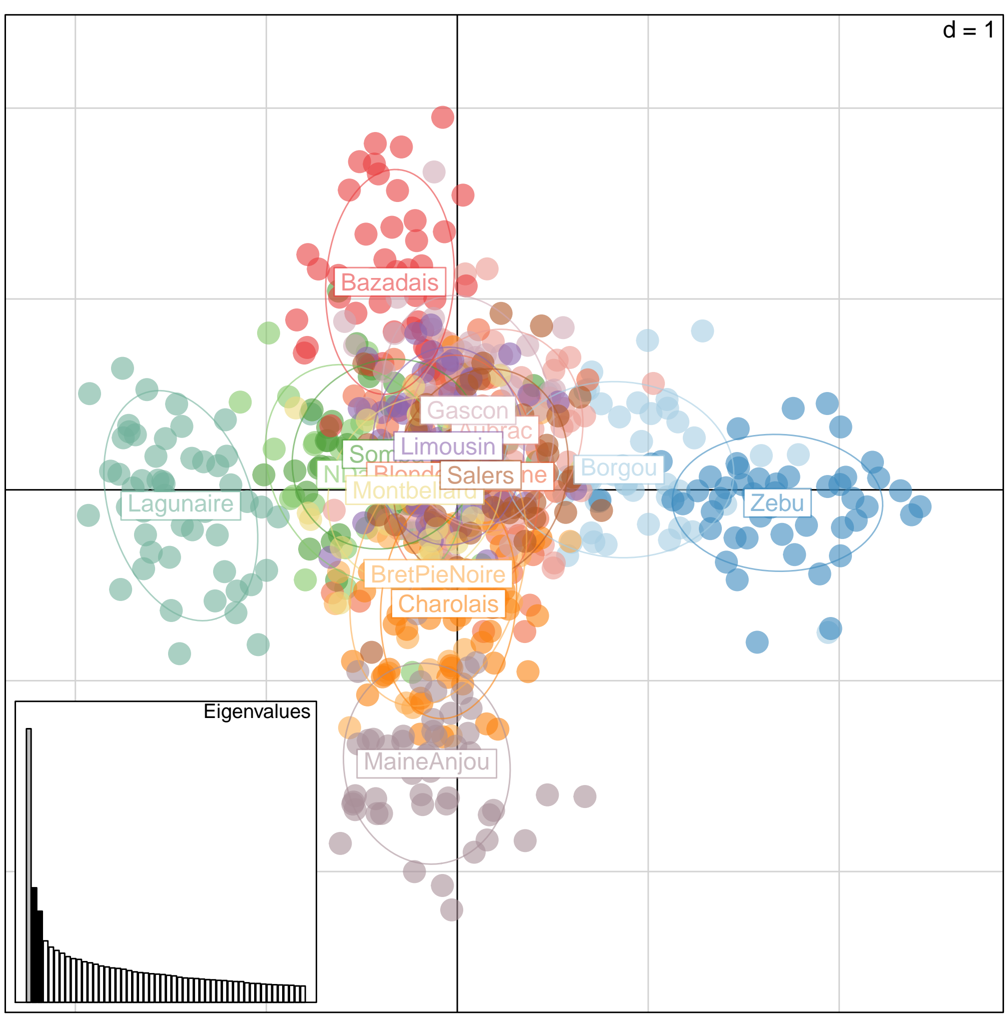
Ellipses indicate the distribution of the individuals from different groups. We can customize this graphic a little, by removing ellipse axes, adding a screeplot of the first 50 eigenvalues in inset, and making colors transparent to better assess overlapping points:

```
s.class(pca.cows$li, fac=pop(microbov),
        col=transp(funky(15),.6),
        axesel=FALSE, cstar=0, cpoint=3)
add.scatter.eig(pca.cows$eig[1:50],3,1,2, ratio=.3)
```



Let us examine the second and third axes:

```
s.class(pca.cows$li, fac=pop(microbov),
        xax=2, yax=3, col=transp(funky(15),.6),
        axesel=FALSE, cstar=0, cpoint=3)
add.scatter.eig(pca.cows$eig[1:50],3,2,3, ratio=.3)
```



What is the major factor of genetic differentiation in these cattle breeds? What is the second one? What is the third one?

In PCA, eigenvalues indicate the variance of the corresponding principal components. 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 can be verified easily:

```
pca.cows$eig[1]

## [1] 1.269978

pc1 <- pca.cows$li[,1]
var(pc1)

## [1] 1.271785
```

```
var(pc1)*703/704

## [1] 1.269978

mean(pc1^2)

## [1] 1.269978

n <- length(pc1)
0.5*mean(dist(pc1)^2)*((n-1)/n)

## [1] 1.269978
```

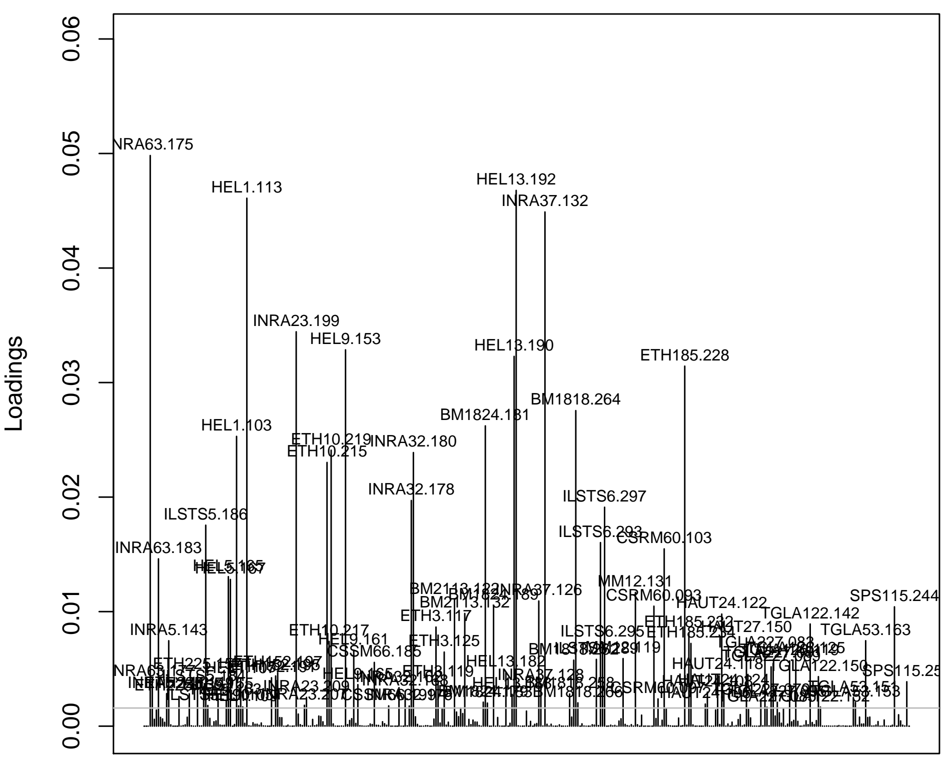
Eigenvalues in `pca.cows$eig` correspond to absolute variances. However, we sometimes want to express these values as percentages of the total variation in the data. This is achieved by a simple standardization:

```
eig.perc <- 100*pca.cows$eig/sum(pca.cows$eig)
head(eig.perc)

## [1] 9.974993 4.176258 3.322746 2.240940 2.014435 1.893127
```

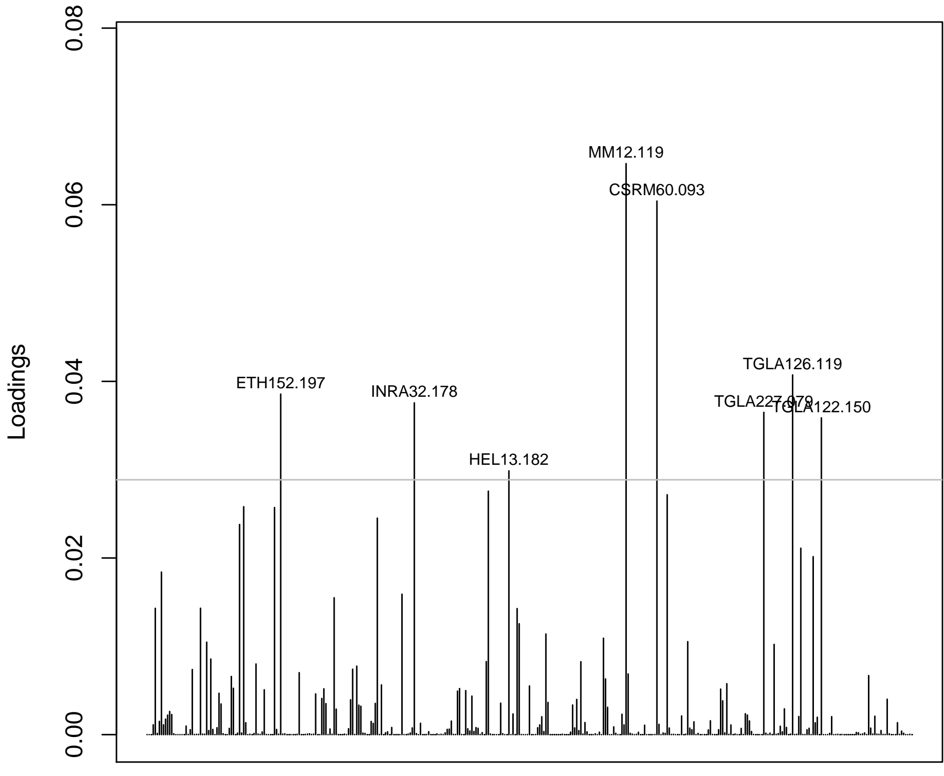
What are the total amounts of variance represented on the plane 1–2 and 2–3?

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 (`pca.cows$c1`); is this informative? An alternative is offered by `loadingplot`, which represents one axis at a time. Interpret the following graph:

```
loadingplot(pca.cows$c1^2)
```



Try using this function to 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"
```

5.2 Principal Coordinates Analysis (PCoA)

Principal Coordinates Analysis (PCoA), also known as Metric Multidimensional Scaling (MDS), is the second most common multivariate analysis in population genetics. This method seeks the best approximation in reduced space of a matrix of Euclidean distances. Its principal components optimize the representation of the squared pairwise distances between individuals. This method is implemented in *ade4* by `dudi.pco`. After scaling the relative allele frequencies of the `microbov` dataset, we perform this analysis:

```
X <- tab(microbov, freq=TRUE, NA.method="mean")
pco.cows <- dudi.pco(dist(X), scannf=FALSE, nf=3)
```

Use `s.class` as before to visualize the results. How are they different from the results of the PCA? What is the meaning of this:

```r
cor(pca.cows$li, pco.cows$li)^2

##                 A1           A2           A3
## Axis1 1.000000e+00 1.017957e-30 1.498668e-31
## Axis2 4.586757e-30 1.000000e+00 5.306284e-30
## Axis3 8.169379e-31 1.012067e-29 1.000000e+00
```

In general, would you recommend using PCA or PCoA to analyse individual data? When would you recommend using PCoA?

6 To go further

More population genetics methods and a more comprehensive list of multivariate methods are presented in the *basics tutorial*, which you can access from the *adegenet* website: http://adegenet.r-forge.r-project.org/

or by typing:

```
adegenetTutorial("basics")
```

For a review of multivariate methods used in genetics:

Jombart *et al.* (2009) Genetic markers in the playground of multivariate analysis. *Heredity* **102**: 330-341. doi:10.1038/hdy.2008.130

For a general, fairly comprehensive introduction to multivariate analysis for ecologists:

Legendre & Legendre (2012) Numerical Ecology, Elsevier