Introduction to population genetics analysis using 

Thibaut Jombart *

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

March 24, 2014

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

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 1.4-1. Make sure you have a recent version of R ($\geq$ 3.0.3) by typing:

```
R.version.string

## [1] "R version 3.0.3 (2014-03-06)"
```

Then, install *adegenet* with dependencies using:

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

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

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

We can now load the package using:

```
library("adegenet")
```

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("Hardy-Weinberg")
```

replies that there is a function `HWE.test.genind` in the *adegenet* package, and 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 and parentheses can be removed):

```
?fstat
```

will open up the help of the function computing F statistics. 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 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 manpage to choose the tutorial to open). 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*: 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 frequencies (i.e. summing to 1 for each locus and each individual)

- `genpop`: allelic data for groups of individuals ("populations") stored as numbers of alleles

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 read a file with GENETIX format which is distributed with the package. Because the path to the file is system-dependent, we use `system.file` to access it:

```
myFile <- system.file("files/nancycats.gtx",package="adegenet")
myFile

## [1] "/usr/local/lib/R/library/adegenet/files/nancycats.gtx"
```

Try and visualize this file using `file.show`. This is the raw input data. To read this file into R and convert it to `genind`, use:

```
cats <- import2genind(myFile)

##
##  Converting data from GENETIX to a genind object...
##
## ...done.
```

Note that this dataset is also distributed as a R object called `nancycats` (loaded using `data(nancycats)`). To know more about these data, see the documentation:

```
?nancycats
```

3 First look at the data

The `genind` object `cats` contains various information:

```
cats

##
##    ####################
##    ### Genind object ###
##    ####################
## - genotypes of individuals -
##
## S4 class:  genind
## @call: read.genetix(file = file, missing = missing, quiet = quiet)
##
## @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: - empty -
```

Data are stored as allele frequencies 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]

##     L1.01 L1.02 L1.03 L1.04 L1.05 L1.06 L1.07 L1.08 L1.09 L1.10 L1.11
## 001    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

```
## 002    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 003     0     0     0     0     0     0     0   0.0   0.5     0     0
## 004     0     0     0     0     0     0     0   0.5   0.5     0     0
## 005     0     0     0     0     0     0     0   0.5   0.5     0     0
##     L1.12 L1.13 L1.14 L1.15 L1.16 L2.01 L2.02 L2.03 L2.04
## 001    NA    NA    NA    NA    NA     0     0     0   0.5
## 002    NA    NA    NA    NA    NA     0     0     0   0.0
## 003     0   0.5     0     0     0     0     0     0   0.5
## 004     0   0.0     0     0     0     0     0     0   0.0
## 005     0   0.0     0     0     0     0     0     0   0.0
```

Most of it is accessible using small functions called "*accessors*":

- `nInd`: returns the number of individuals in the object.
- `nLoc`: returns the number of loci.
- `indNames`†: returns/sets individual labels.
- `locNames`†: returns/sets labels of the loci.
- `alleles`†: returns/sets alleles.
- `ploidy`†: returns/sets ploidy of the individuals.
- `pop`†: returns/sets a factor grouping individuals.
- `other`†: returns/sets misc information stored as a list.

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

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

##    001    002    003    004    005    006    007    008    009    010
## "N215" "N216" "N217" "N218" "N219" "N220" "N221" "N222" "N223" "N224"

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

##      001      002      003      004      005      006      007      008
##  "cat.1"  "cat.2"  "cat.3"  "cat.4"  "cat.5"  "cat.6"  "cat.7"  "cat.8"
##      009      010
##  "cat.9" "cat.10"
```

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

```
locNames(nancycats)

## Error:  error in evaluating the argument 'x' in selecting a method for
function 'locNames':  Error:  object 'nancycats' not found
```

returns the names of the loci, while:

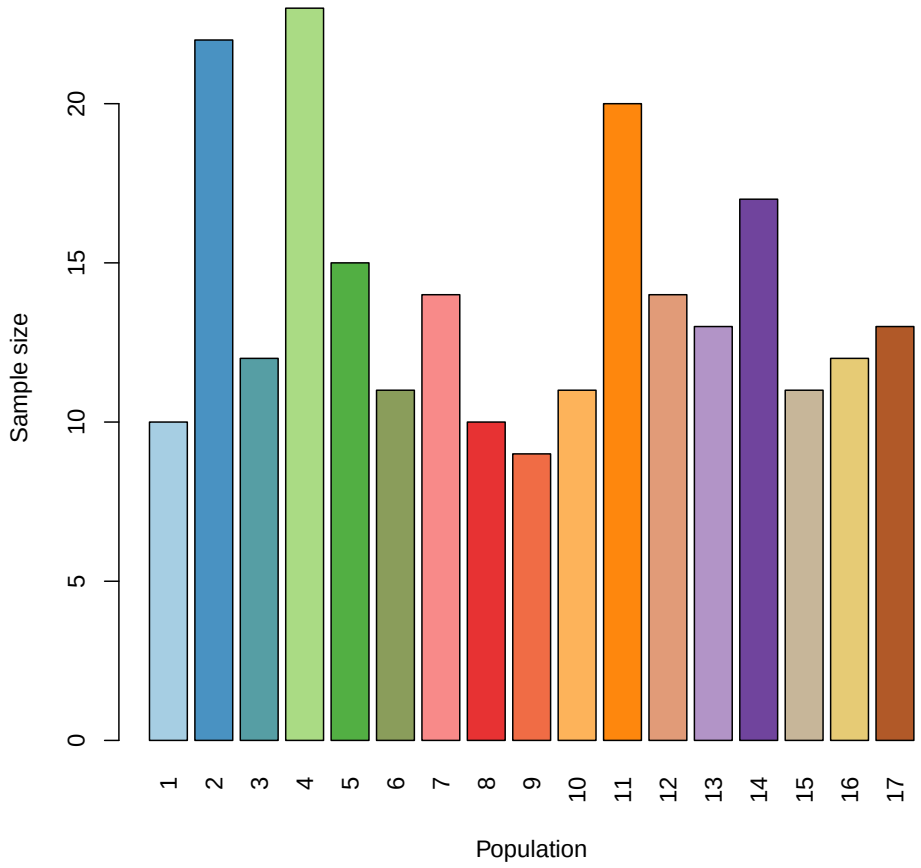
```
temp <- locNames(nancycats, withAlleles=TRUE)

## Error:  error in evaluating the argument 'x' in selecting a method for
function 'locNames':  Error:  object 'nancycats' not found

head(temp, 10)

##          1       2       3       4       5       6       7       8       9
## 1       NA 0.08018 0.07141 0.04993 0.05907 0.07820 0.06123 0.05778 0.08784
## 2  0.08018      NA 0.08201 0.06985 0.08295 0.06406 0.07047 0.05572 0.07205
## 3  0.07141 0.08201      NA 0.02572 0.05305 0.03826 0.07606 0.05839 0.08049
## 4  0.04993 0.06985 0.02572      NA 0.03834 0.03450 0.05523 0.03450 0.04839
## 5  0.05907 0.08295 0.05305 0.03834      NA 0.06068 0.08426 0.06662 0.08653
## 6  0.07820 0.06406 0.03826 0.03450 0.06068      NA 0.06388 0.05954 0.07889
## 7  0.06123 0.07047 0.07606 0.05523 0.08426 0.06388      NA 0.04758 0.09919
## 8  0.05778 0.05572 0.05839 0.03450 0.06662 0.05954 0.04758      NA 0.08853
## 9  0.08784 0.07205 0.08049 0.04839 0.08653 0.07889 0.09919 0.08853      NA
## 10 0.05726 0.05314 0.07568 0.06228 0.06968 0.07047 0.07708 0.05531 0.08022
##         10      11      12      13      14      15      16
## 1  0.05726 0.04199 0.06472 0.07631 0.05635 0.06157 0.08898
## 2  0.05314 0.06011 0.07615 0.08019 0.06752 0.05951 0.07135
## 3  0.07568 0.03006 0.06599 0.08200 0.04061 0.05610 0.07781
## 4  0.06228 0.02650 0.04369 0.06439 0.03424 0.04021 0.05262
## 5  0.06968 0.04531 0.06805 0.08075 0.05610 0.07191 0.07062
## 6  0.07047 0.02928 0.06757 0.07048 0.04941 0.06724 0.05317
## 7  0.07708 0.05446 0.06336 0.08972 0.07526 0.07865 0.08940
## 8  0.05531 0.04249 0.06426 0.07841 0.04260 0.04885 0.08104
## 9  0.08022 0.04603 0.06610 0.10242 0.05293 0.06305 0.07184
## 10      NA 0.04051 0.08141 0.08902 0.05988 0.05030 0.08053
```

returns the names of the alleles in the form "`loci.allele`".

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

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

##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
## [36] 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

```
table(pop(cats))

##
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## 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")
```

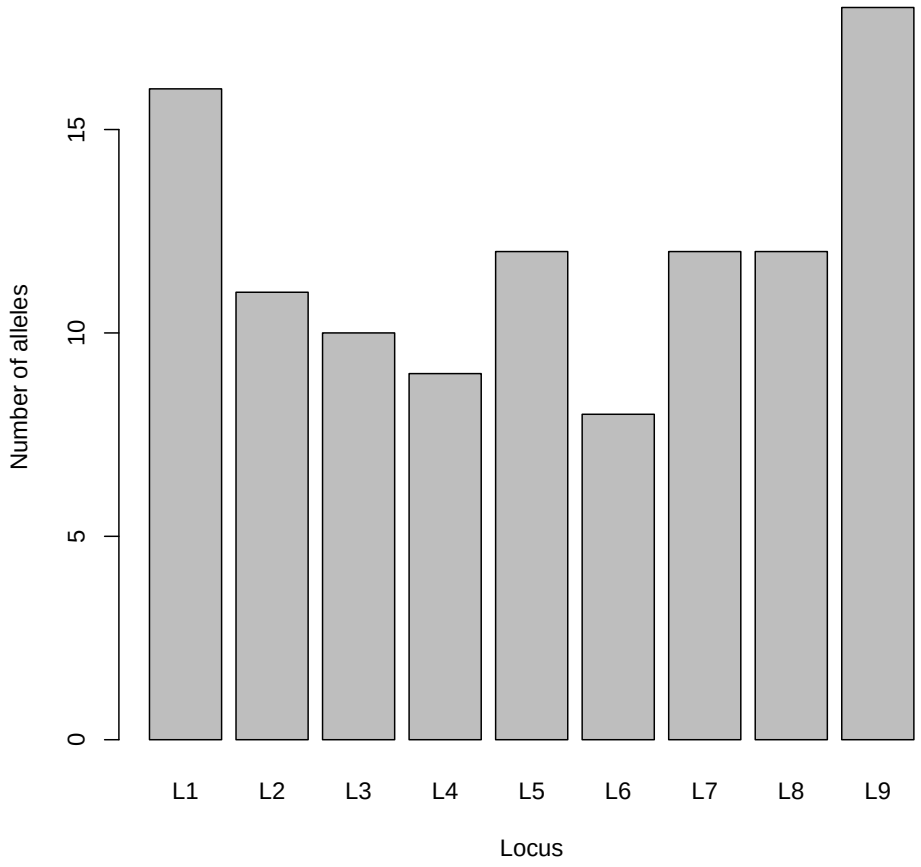


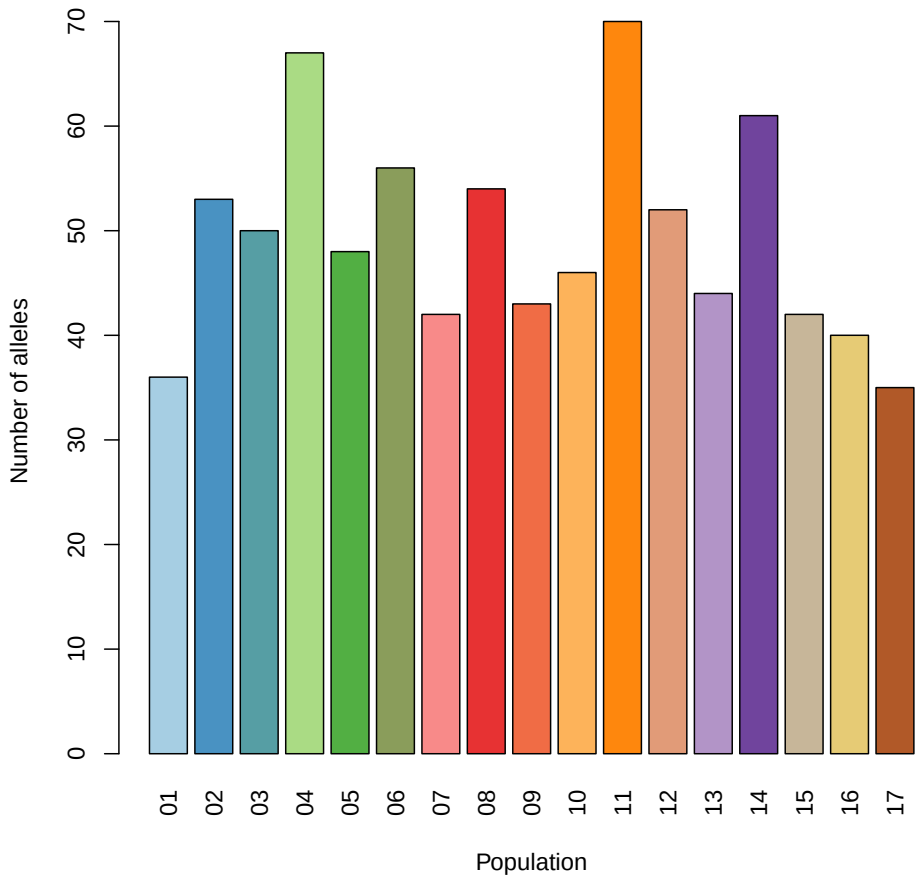
More information is available from the `summary`:

```
temp <- summary(cats)

##
##  # Total number of genotypes:  237
##
##  # Population sample sizes:
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## 10 22 12 23 15 11 14 10  9 11 20 14 13 17 11 12 13
```

```
##
##  # Number of alleles per locus:
## L1 L2 L3 L4 L5 L6 L7 L8 L9
## 16 11 10  9 12  8 12 12 18
##
##  # Number of alleles per population:
## 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
## 36 53 50 67 48 56 42 54 43 46 70 52 44 61 42 40 35
##
##  # Percentage of missing data:
## [1] 2.344
##
##  # Observed heterozygosity:
##     L1     L2     L3     L4     L5     L6     L7     L8     L9
## 0.6682 0.6667 0.6793 0.7083 0.6329 0.5654 0.6498 0.6184 0.4515
##
##  # Expected heterozygosity:
##     L1     L2     L3     L4     L5     L6     L7     L8     L9
## 0.8657 0.7929 0.7953 0.7603 0.8703 0.6885 0.8158 0.7603 0.6063
```

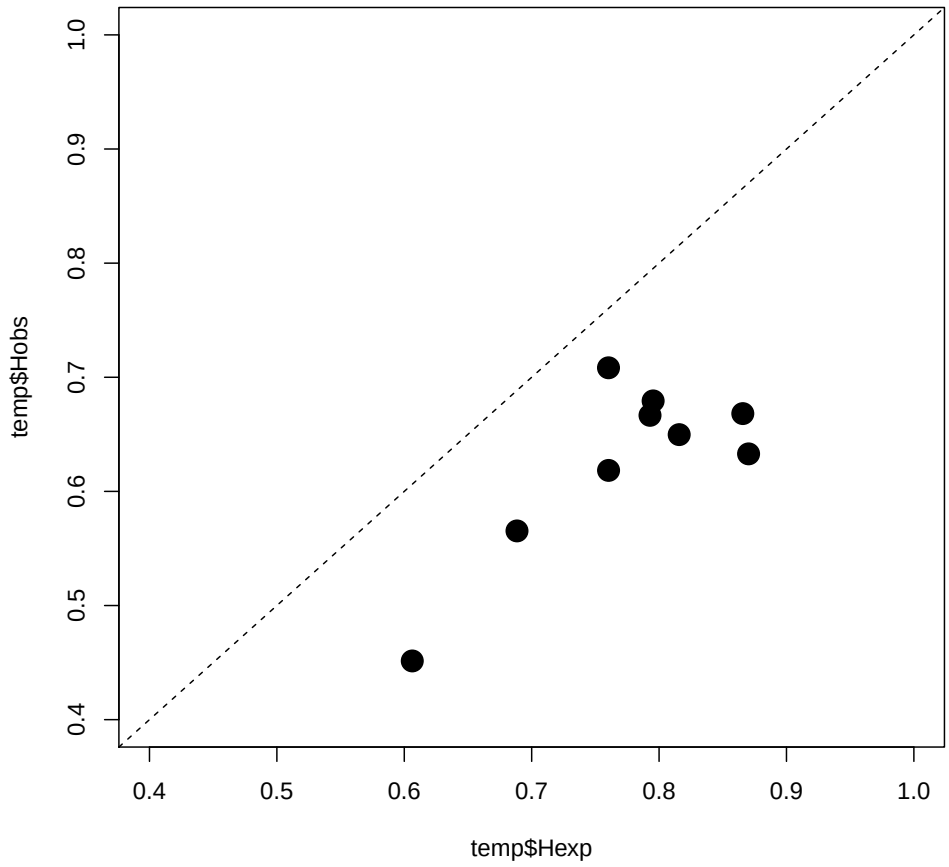
`temp` contains the information returned by `summary`. Using the same function as above, try displaying the number of alleles i) per locus and ii) per population. You should obtain something along the lines of:


Number of alleles
0
5
10
15
L1
L2
L3
L4
L5
L6
L7
L8
L9
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

The Hardy-Weinberg equilibrium (HWE) test is implemented for `genind` objects. The function to use is `HWE.test.genind`, and requires the package *genetics*. Here we first produce a matrix of p-values (`res="matrix"`) using parametric test. Monte Carlo procedure are more reliable but also more computer-intensive (use `permut=TRUE`).

```
allpval <- HWE.test.genind(nancycats,res="matrix")

## Error:  object 'nancycats' not found

dim(allpval)
```

```
## Error:  object 'allpval' not found
```

One test is performed per locus and population, *i.e.* 153 tests in this case. Thus, the first question is: which tests are highly significant?

```
colnames(allpval)

## Error:  object 'allpval' not found

idx <- which(allpval<0.0001,TRUE)

## Error:  object 'allpval' not found

idx

## Error:  object 'idx' not found
```

`idx` indicates the indices of the significant outcomes (populations in rows, loci in columns). We can seek complete tests for these combinations only:

```
alltests <- HWE.test.genind(nancycats,res="full")

## Error:  object 'nancycats' not found

mapply(function(i,j) alltests[[i]][[j]], idx[,2], idx[,1], SIMPLIFY=FALSE)

## Error:  object 'idx' not found
```

What is your conclusion concerning HWE in these data?

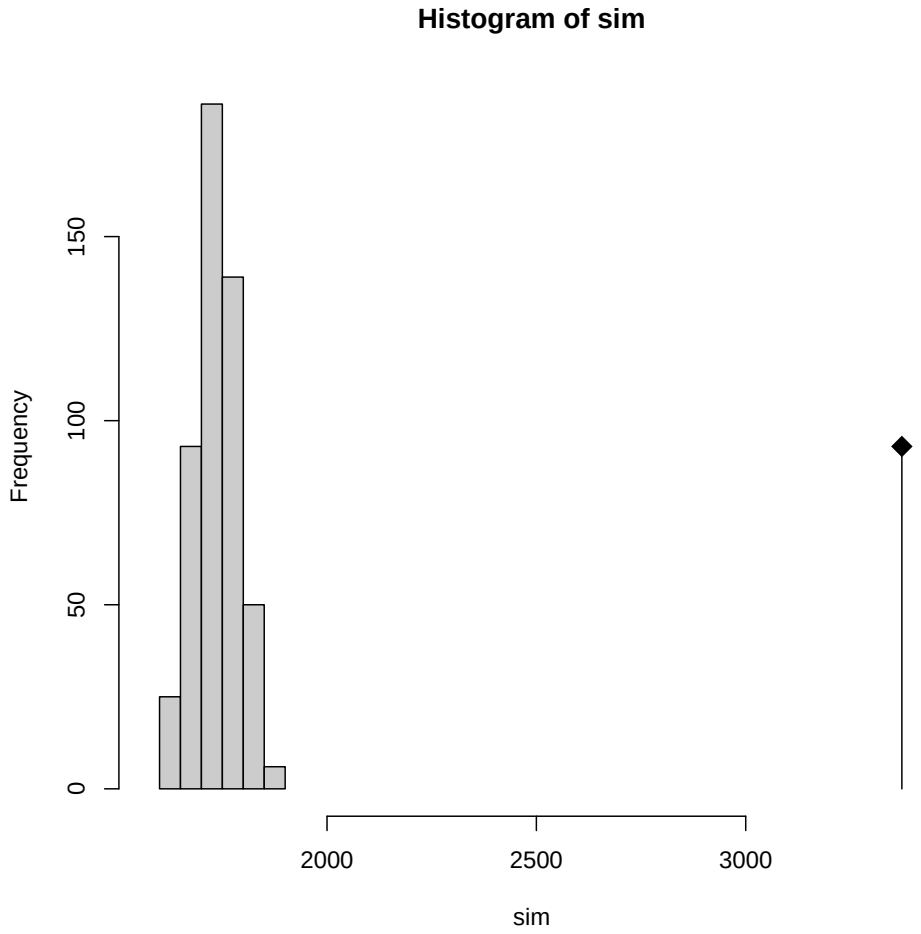
4.2 Assessing population structure

Population structure is traditionally measured and tested using F statistics, in particular the *Fst*, 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, the most simple of which have wrappers in *adegenet*.

Here, we first test the overall population structure using `gstat.randtest`. Try to interpret the following statistics and graphics:

```
fstat(nancycats, fstonly=TRUE)

## [1] 0.08495

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

```
## Monte-Carlo test
## Call: gstat.randtest(x = cats)
##
## Observation: 3373
##
## Based on 499 replicates
## Simulated p-value: 0.002
## Alternative hypothesis: greater
##
##     Std.Obs Expectation    Variance
##       32.52     1735.87     2533.90

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 Fst values between pairs of populations. This can be done using the function `pairwise.fst`, which computes Nei's

estimator of pairwise Fst defined as:

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

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:

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

##         1       2       3       4
## 1 0.00000 0.08018 0.07141 0.04993
## 2 0.08018 0.00000 0.08201 0.06985
## 3 0.07141 0.08201 0.00000 0.02572
## 4 0.04993 0.06985 0.02572 0.00000
```

We remove population 17 for which computations were not possible:

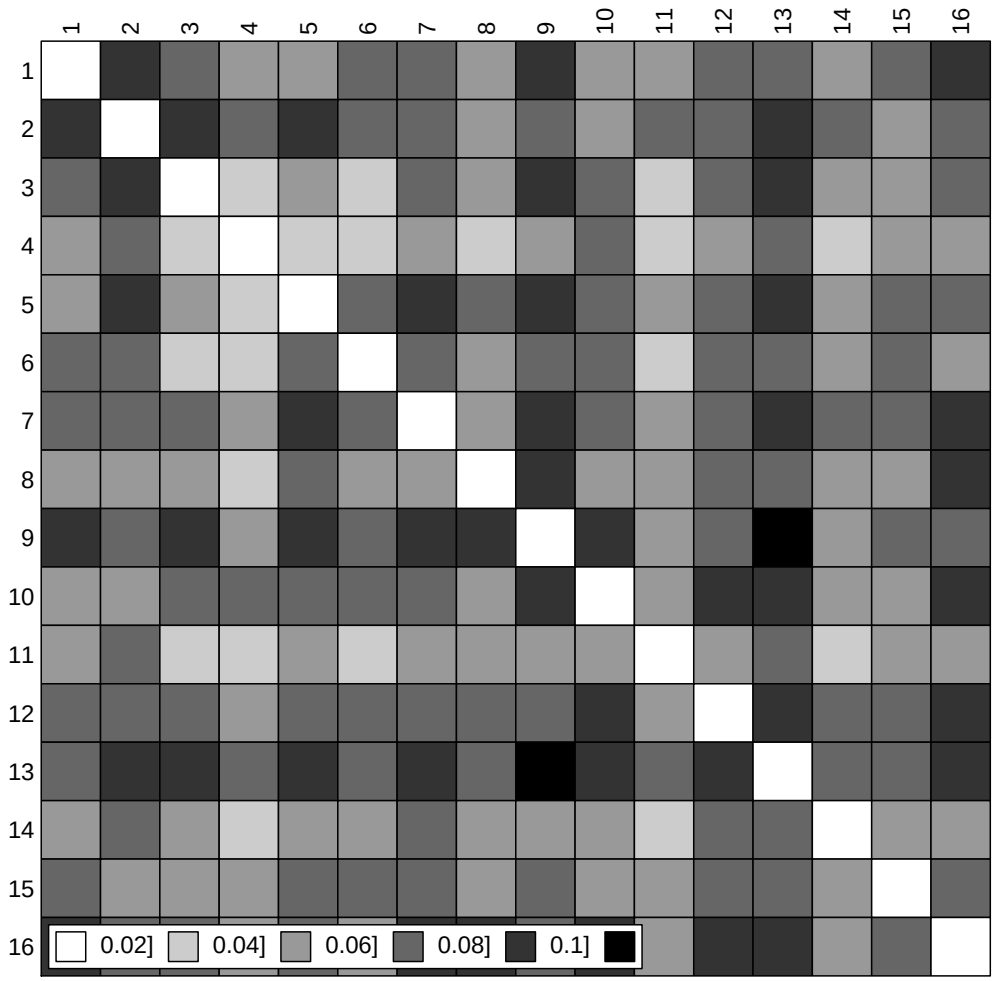
```
matFst <- matFst[-17,-17]
```

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(matFst)

## Error:  could not find function "nj"

plot(cats.tree, type="unr", tip.col=funky(17)[-17], font=2)

## Error:  error in evaluating the argument 'x' in selecting a method for
function 'plot':  Error:  object 'cats.tree' not found

annot <- round(cats.tree$edge.length,2)

## Error:  object 'cats.tree' not found

edgelabels(annot[annot>0], which(annot>0), frame="n")

## Error:  could not find function "edgelabels"

add.scale.bar()

## Error:  could not find function "add.scale.bar"
```

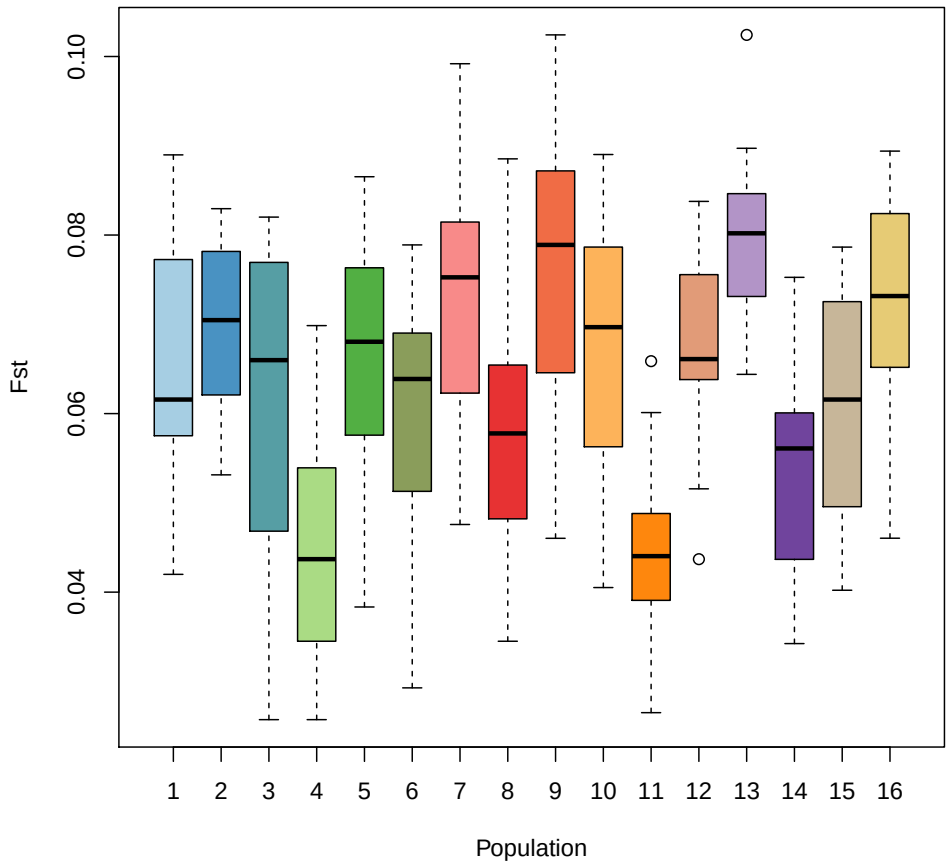
What can you say about the population structure? Is there an outlying group? To confirm your intuition, visualize the raw data using:

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



Interpret the following this figure:

```
temp <- matFst
diag(temp) <- NA
boxplot(temp, col=funky(17)[-17], xlab="Population", ylab="Fst")
```


Fst
0.04
0.06
0.08
0.10
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Population

5 Multivariate analyses

5.1 Principal Component Analysis (PCA)

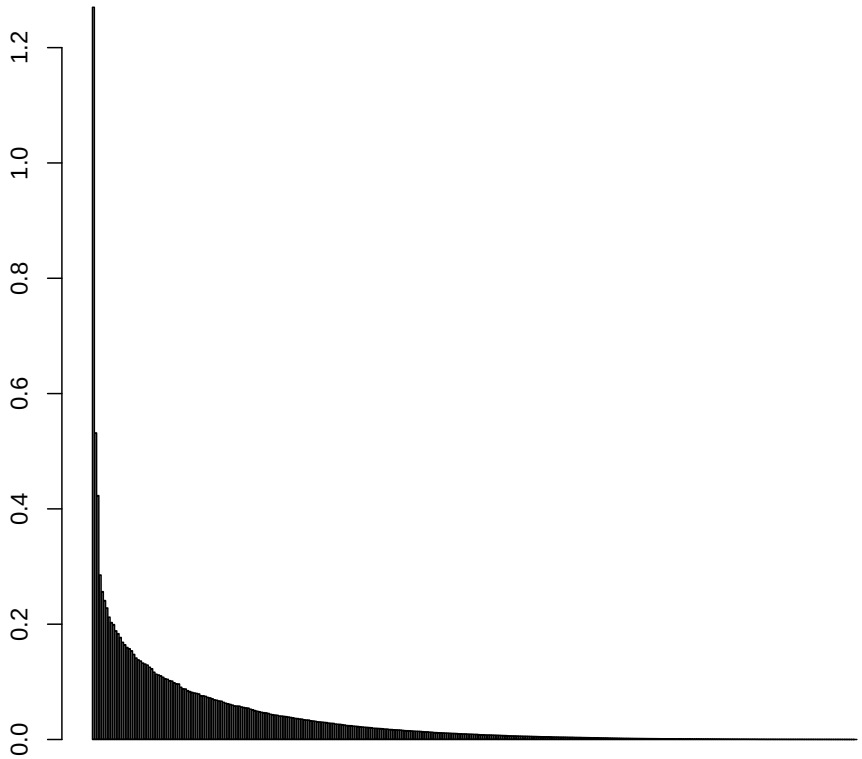
Principal Component Analysis (PCA) is the amongst the most common multivariate analyses used in genetics. Running a PCA on `genind` object is straightforward. One needs to first scale the data and replace missing data (`scaleGen`) and then use the PCA procedure (`dudi.pca`). Let us use this approach on the `microbov` data, which contain a set of 30 microsatellites sampled for 704 cows from 15 breeds (see `?microbov`). Let us first load the data:

```
data(microbov)
```

Note that as we use `scaleGen` to do the centring/scaling of the data, we disable these in `dudi.pca`:

```
x.cows <- scaleGen(microbov, missing="mean", scale=FALSE)
```

```
pca.cows <- dudi.pca(x.cows, center=FALSE, 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?

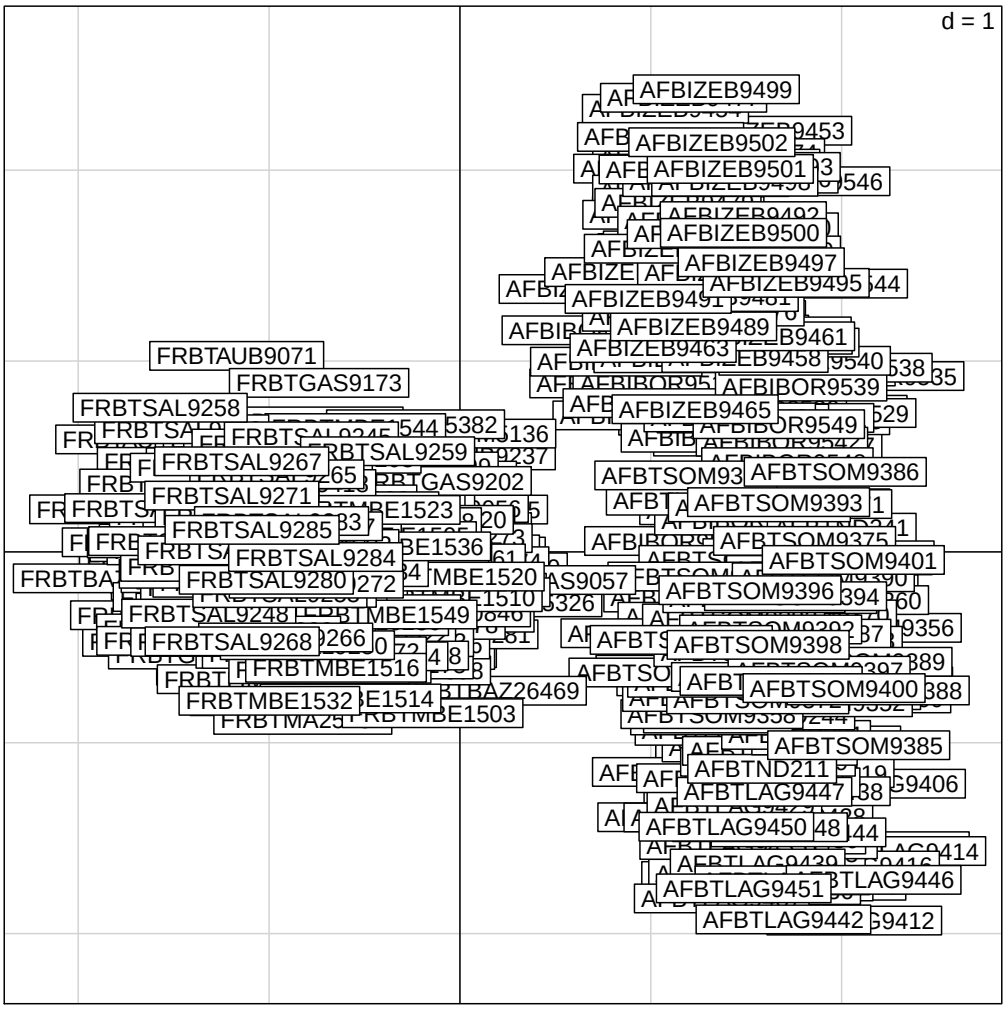
```
pca.cows

## Duality diagramm
## class: pca dudi
## $call: dudi.pca(df = x.cows, center = 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 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.

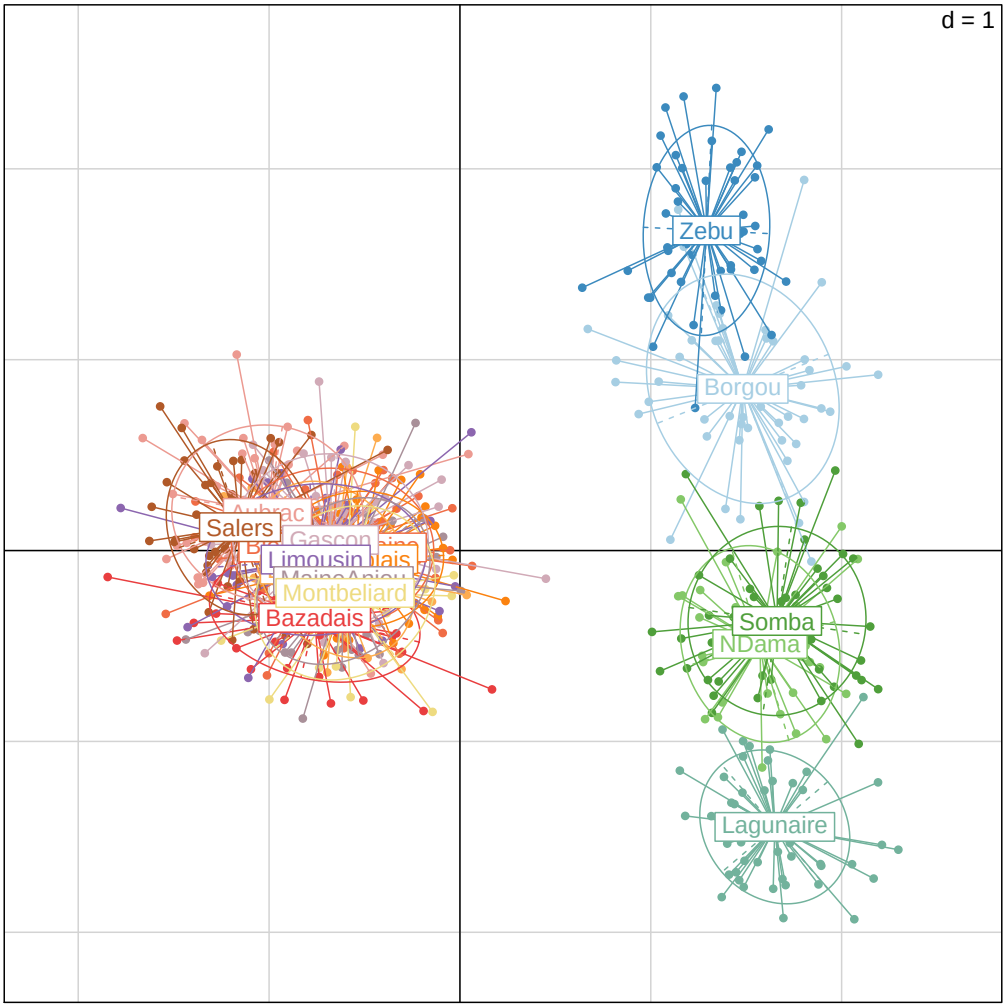
Coordinates of individual genotypes onto the principal axes can be visualized using `s.label`:

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



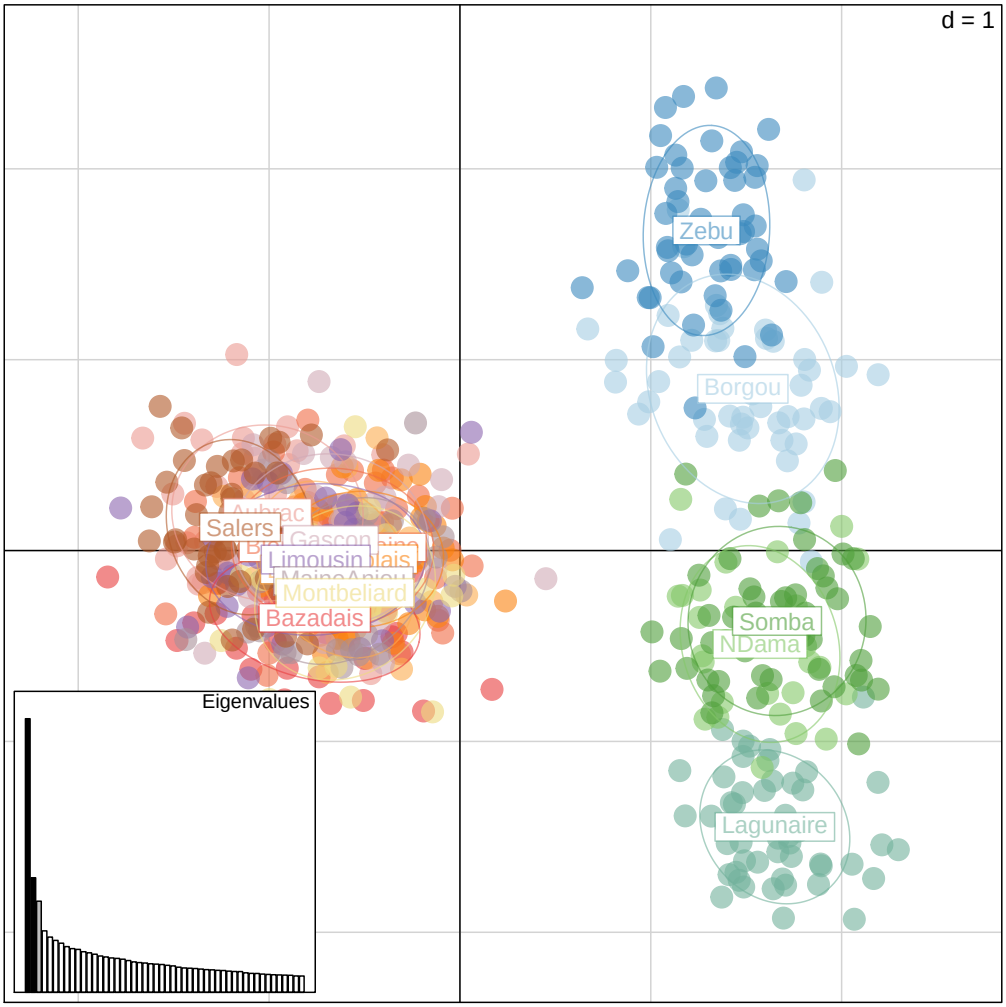
This is, however, not very usefull here. Let us add group information using `s.class`:

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



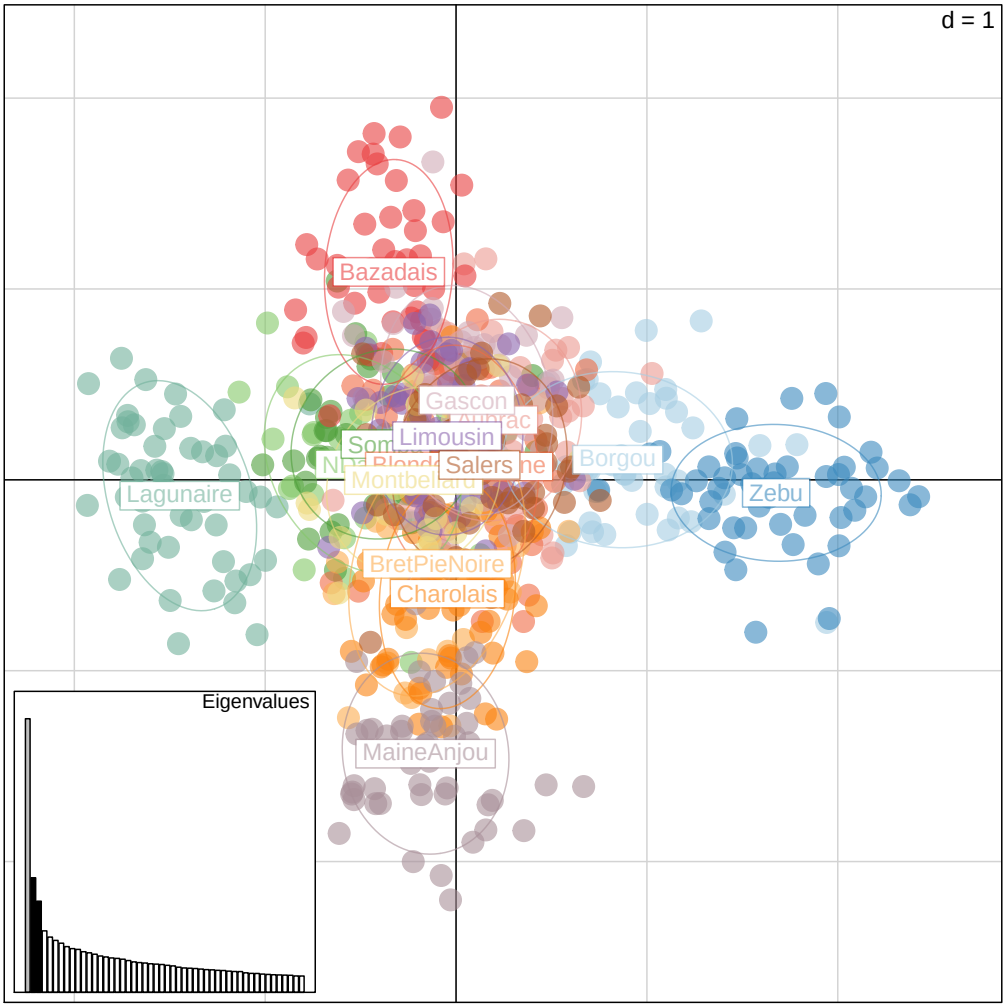
Ellipses indicate the distribution of the individuals from different groups. We can customize a little this graphic, by removing ellipse axes, adding a screeplot of the first 50 eigenvalues in inset, and making colors transparent to better assess overlapping points:

```
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, 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 can be verified easily:

```
pca.cows$eig[1]
```

```
## [1] 1.27
```

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

```
## [1] 1.27
```

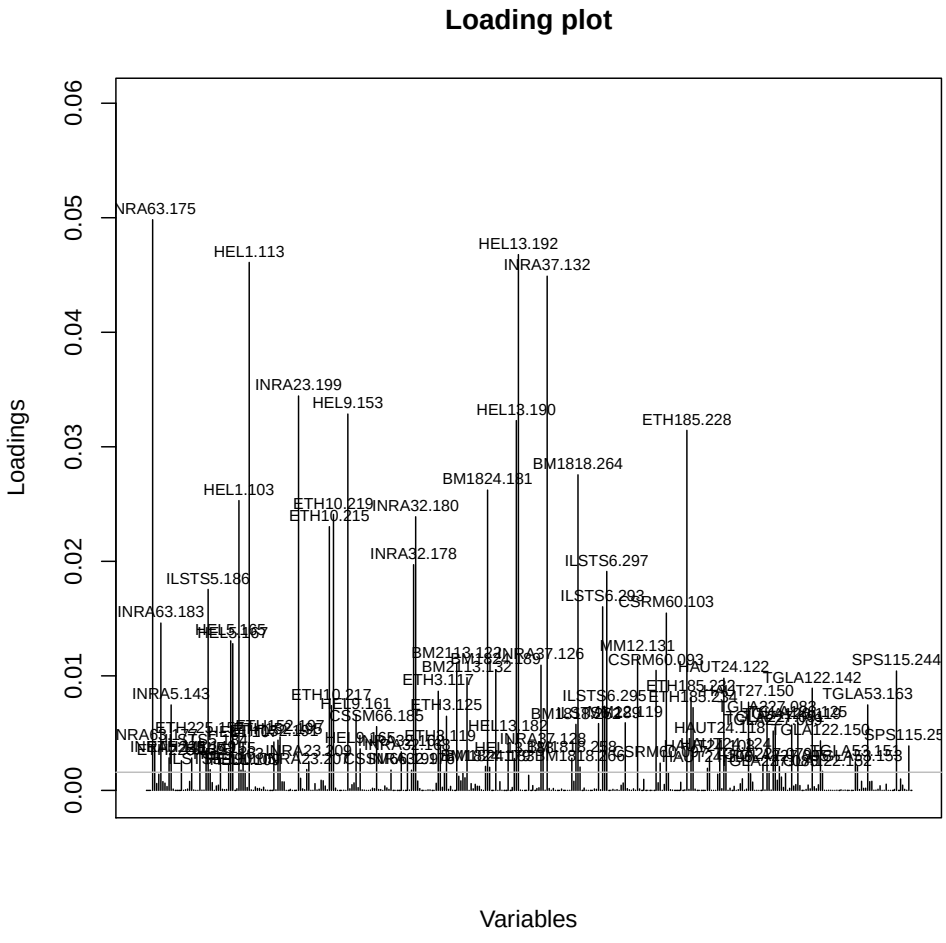
Eigenvalues in `pca.cows$eig` correspond to absolute variances. However, we sometimes want to express these values as percentages of the total variation in the data. This is achieved by a simple standardization:

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

## [1] 9.975 4.176 3.323 2.241 2.014 1.893
```

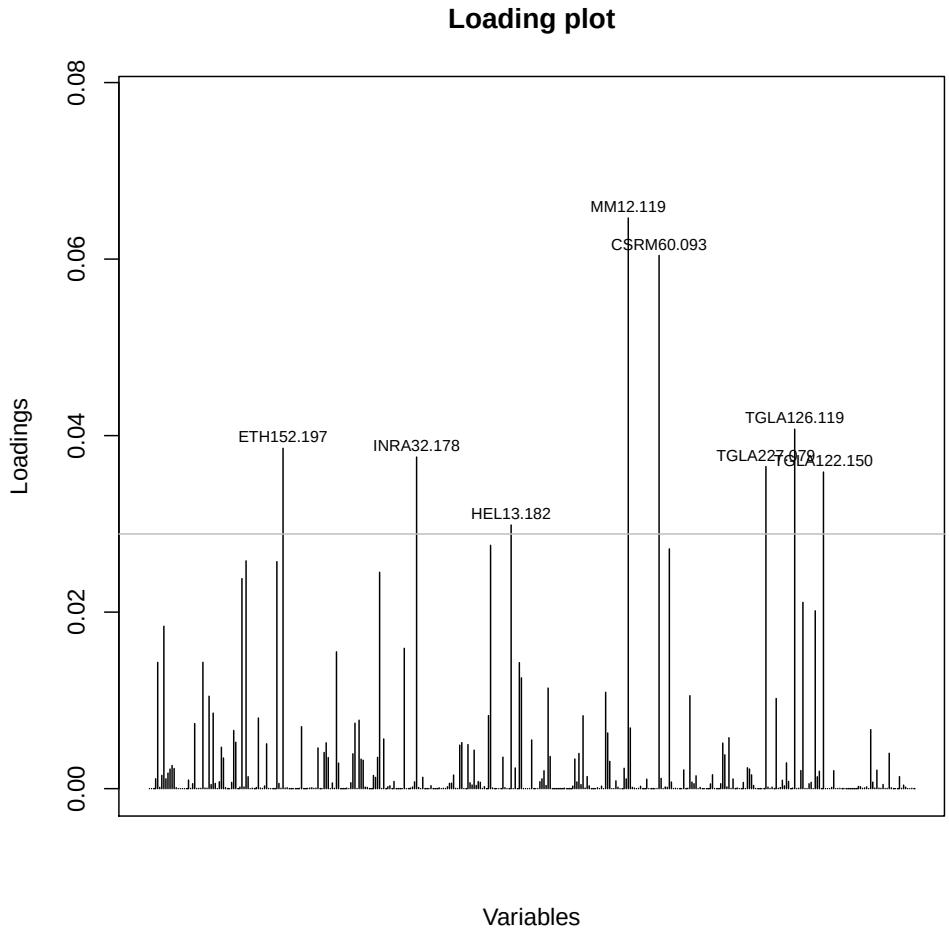
What are the total amounts of variance represented on the plane 1–2 and 2–3?

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 <- scaleGen(microbov, scale=FALSE, miss="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 1.000e+00 2.054e-30 1.049e-31
## Axis2 5.497e-30 1.000e+00 1.041e-29
## Axis3 6.245e-31 9.076e-30 1.000e+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