Tutorial using the free software 

A tutorial for the R package `adegenet`

T. JOMBART

April 2006 - Version 1.0

For any questions or comments, please send an email to:
`jombart@biomserv.univ-lyon1.fr`.

Contents

1 Introduction

This tutorial proposes a short visit through functionalities of the `adegenet` package for R (Ihaka & Gentleman, 1996; R Development Core Team, 2006). The purpose of this package is to facilitate the multivariate analysis of molecular marker data, especially using the `ade4` package (Chessel *et al.*, 2004). Data can be imported from popular softwares like GENETIX, or converted from simple data frame of genotypes. `adegenet` also aims at providing a platform from which to use easily methods provided by other R packages (e.g., Goudet, 2005). Indeed, if it is possible to perform various genetic data analyses using R, data formats often differ from one package to another, and conversions are sometimes far from easy and straightforward.

In this tutorial, I first present the two object classes used in `adegenet`, namely `genind` (genotypes of individuals) and `genpop` (genotypes grouped by populations). Then, several topics will be tackled using reproductible examples.

2 First steps

2.1 Installing the package

Current version of the package is 1.0, and is compatible with R 2.4.1 and 2.5. Here the `adegenet` package is installed along with other recommended packages.

```
> install.packages("adegenet", dep = TRUE)
> install.packages("ade4", dep = TRUE)
> install.packages("hierfstat", dep = TRUE)
> install.packages("genetics", dep = TRUE)
```

Then the first step is to load the package:

```
> library(adegenet)
```

2.2 Object classes

Two classes of objects are defined, depending on the scale at which the genetic information is stored: `genind` is used for individual genotypes, whereas `genpop` is used for alleles numbers counted by populations. Note that the term 'population', here and later, is employed in a broad sense: it simply refers to any grouping of individuals.

2.2.1 genind objects

These objects can be obtained by importation from foreign softwares or by conversion from a table of allelic frequencies (see 'importing data').

```
> data(nancycats)
> is.genind(nancycats)


[1] TRUE


> nancycats


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

class: [1] "genind"

$call: [1] NA

$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

Optionnal contents:
$pop: factor giving the population of each individual
$pop.names: vector giving the names of the populations

other elements: $xy
```

A `genind` object is a list of several components (see ?genind). The main one is a table of allelic frequencies of individuals (in rows) for every alleles in every loci. Being frequencies, data sum to one per locus, giving the score of 1 for an homozygote and 0.5 for an heterozygote. For instance:

```
> nancycats$tab[10:18, 1:10]


    L1.01 L1.02 L1.03 L1.04 L1.05 L1.06 L1.07 L1.08 L1.09 L1.10
010     0     0     0     0     0   0.0   0.0   0.0   1.0   0.0
011     0     0     0     0     0   0.0   0.0   0.0   0.0   0.5
012     0     0     0     0     0   0.5   0.0   0.5   0.0   0.0
013     0     0     0     0     0   0.5   0.0   0.5   0.0   0.0
014     0     0     0     0     0   0.0   0.0   1.0   0.0   0.0
015     0     0     0     0     0   0.0   0.5   0.0   0.5   0.0
016     0     0     0     0     0   0.5   0.0   0.0   0.5   0.0
017     0     0     0     0     0   0.5   0.0   0.5   0.0   0.0
018     0     0     0     0     0   0.5   0.0   0.0   0.5   0.0
```

Individual '010' is an homozygote for the allele 09 at locus 1, while '018' is an heterozygote with alleles 06 and 09. For an homogeneity purpose, generic labels are given to individuals, locus, alleles and eventually population. The true names are stored in the object (components `$[...].names` where ... can be 'ind', 'loc', 'all' or 'pop'). For instance :

```
> nancycats$loc.names
```

```
     L1      L2      L3      L4      L5      L6      L7      L8      L9
 "fca8" "fca23" "fca43" "fca45" "fca77" "fca78" "fca90" "fca96" "fca37"
```

gives the true marker names, and

```
> nancycats$all.names[[3]]
```

```
   01    02    03    04    05    06    07    08    09    10
"133" "135" "137" "139" "141" "143" "145" "147" "149" "157"
```

gives the allele names for marker 3.

Note that optional components are also allowed, but will not be given generic names. The component $pop (a factor giving a grouping of individuals) is particular in that the behaviour of many functions will check automatically for it and adapt accordingly. In fact, each time an argument 'pop' is required by a function, it is first seeked in the object. For instance, using the function `genind2genpop` to convert `nancycats` to a `genpop` object, there is no need to give a 'pop' argument as it exists in the `genind` object:

```
> table(nancycats$pop)
```

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

```
> catpop <- genind2genpop(nancycats)
```

```
 Converting data from a genind to a genpop object...

...done.
```

Other additional components can be stored (like here, spatial coordinates of populations in $xy) but will not be passed during any conversion (`catpop` has no $xy).

Finally, a `genind` object generally contains its matched call, *i.e.* the instruction that created itself. This is not the case, however, for objects loaded using `data`. When call is available, it can be used to regenerate an object.

```
> nancycats$call

[1] NA

> obj <- genetix2genind(system.file("files/nancycats.gtx", package = "adegenet"))

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

> obj$call

genetix2genind(file = system.file("files/nancycats.gtx", package = "adegenet"))

> toto <- eval(obj$call)

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

> identical(obj, toto)

[1] TRUE
```

2.2.2 genpop objects

We use the previously built genpop object:

```
> catpop

       ####################
       ### Genpop object ###
       ####################
- Alleles counts for populations -

class: [1] "genpop"

$call: genind2genpop(x = nancycats)

$tab:  17 x 108 matrix of alleles counts

$pop.names: vector of  17 population 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

other elements: NULL
```

```
> is.genpop(catpop)

[1] TRUE

> catpop$tab[1:5, 1:10]

    L1.01 L1.02 L1.03 L1.04 L1.05 L1.06 L1.07 L1.08 L1.09 L1.10
P01     0     0     0     0     0     0     0     2     9     1
P02     0     0     0     2     0     0     0     0    10     0
P03     0     0     0     7     0     0     0     0     7     1
P04     0     0     0     2     0     5     0     0     3     3
P05     0     0     0     0     0     5    13     2     3     3
```

The matrix \$tab contains alleles counts per population (here, cat colonies). These objects are otherwise very similar to `genind` in their structure, and possess generic names, true names, and the matched call.

3 Various topics

3.1 Importing data

Presently, data importation is available from GENETIX (.gtx), Fstat (.dat) and Genepop (.gen) files. Note that Easypop data simulation software provides both .dat and .gen files. In all cases, only `genind` will be produced. The associated functions are `genetix2genind`, `fstat2genind` and `genepop2genind`. For simplification, one can use the generic function `import2genind` which detects a file format from its extension and uses the appropriate routine. For instance:

```
> obj1 <- genetix2genind(system.file("files/nancycats.gtx", package = "adegenet"))

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

> obj2 <- import2genind(system.file("files/nancycats.gtx", package = "adegenet"))

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

> obj1
```

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

class: [1] "genind"

$call: genetix2genind(file = system.file("files/nancycats.gtx", package = "adegenet"))

$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

Optionnal contents:
$pop: factor giving the population of each individual
$pop.names: vector giving the names of the populations

other elements: NULL


> obj2


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

class: [1] "genind"

$call: genetix2genind(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

Optionnal contents:
$pop: factor giving the population of each individual
$pop.names: vector giving the names of the populations

other elements: NULL
```

However, it happens that data are available in other formats. Generally, these can be expressed as a individuals x markers table where each element is a character string coding 2 alleles. Such data are interpretable when all strings contain 2,4 or 6 characters. For instance, "11" will be an homozygote 1/1, "1209" will be an heterozygote 12/09. The function `genetix2genind` can convert such data.

```
> args(genetix2genind)
```

```
function (file = NULL, X = NULL, pop = NULL, missing = NA, quiet = FALSE)
NULL
```

In such case, the X argument is used after reading the data using `read.table`. Here I provide an example using a data set from the library hierfstat.

```
> library(hierfstat)
> toto <- read.fstat.data(paste(.path.package("hierfstat"), "/data/diploid.dat",
+     sep = "", collapse = ""), nloc = 5)
> head(toto)
```

```
  Pop loc-1 loc-2 loc-3 loc-4 loc-5
1   1    44    43    43    33    44
2   1    44    44    43    33    44
3   1    44    44    43    43    44
4   1    44    44    NA    33    44
5   1    44    44    24    34    44
6   1    44    44    NA    43    44
```

`toto` is a data frame containing genotypes and a population factor.

```
> obj <- genetix2genind(X = toto[, -1], pop = toto[, 1])
```

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

...done.
```

```
> obj
```

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

class: [1] "genind"

$call: genetix2genind(X = toto[, -1], pop = toto[, 1])

$tab:  44 x 11 matrix of genotypes

$ind.names: vector of  44 individual names
$loc.names: vector of  5 locus names
$loc.nall: number of alleles per locus
$loc.fac: locus factor for the  11 columns of $tab
$all.names: list of  5 components yielding allele names for each locus

Optionnal contents:
$pop: factor giving the population of each individual
$pop.names: vector giving the names of the populations

other elements: NULL
```

Lastly, `genind` or `genpop` objects can be obtained from a data matrix similar to the `$tab` component (respectively, alleles frequencies and alleles counts). Such action is achieved by `as.genind` and `as.genpop`. Here an example is provided for `as.genpop` using dataset from ade4:

```
> library(ade4)
> data(microsatt)
> microsatt$tab[10:15, 12:15]
```

```
           INRA32.168 INRA32.170 INRA32.174 INRA32.176
Mtbeliard           0          0          0          1
NDama               0          0          0         12
Normand             1          0          0          2
Parthenais          8          5          0          3
Somba               0          0          0         20
Vosgienne           2          0          0          0
```

`microsatt$tab` contains alleles counts, and can therefore be used to make a `genpop` object.

```
> toto <- as.genpop(microsatt$tab)
> toto
```

```
       ####################
       ### Genpop object ###
       ####################
- Alleles counts for populations -

class: [1] "genpop"

$call: as.genpop(tab = microsatt$tab)

$tab:  18 x 112 matrix of alleles counts

$pop.names: vector of  18 population names
$loc.names: vector of  9 locus names
$loc.nall: number of alleles per locus
$loc.fac: locus factor for the  112 columns of $tab
$all.names: list of  9 components yielding allele names for each locus

other elements: NULL
```

3.2 Using summaries

Both `genind` and `genpop` objects have a summary providing basic information about data. Informations are both printed and invisibly returned as a list.

```
> toto <- summary(nancycats)
```

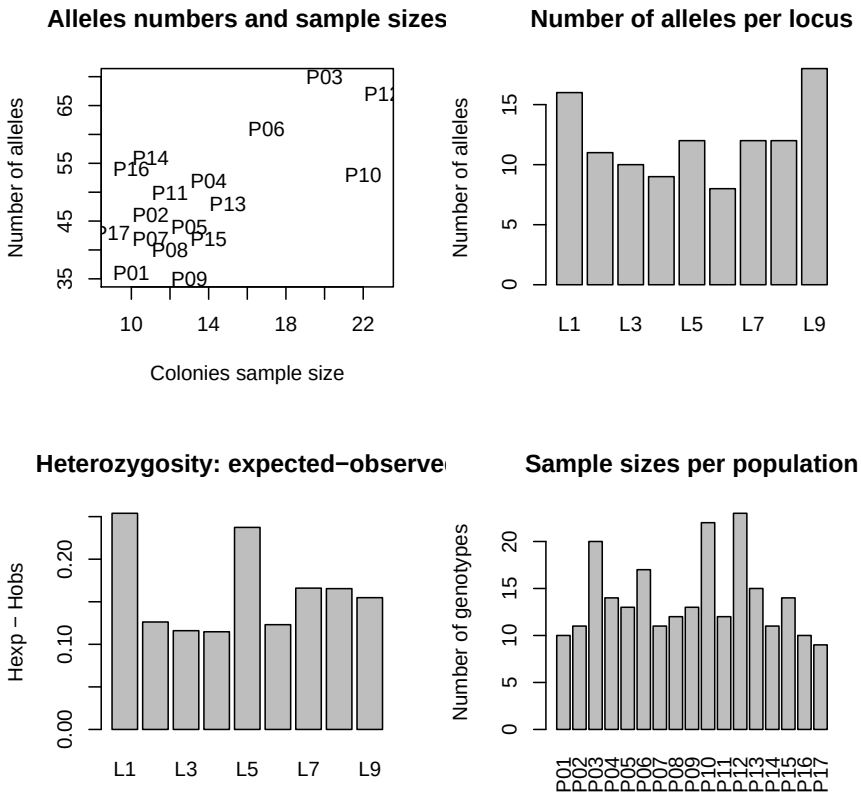
```
 # Total number of genotypes:  237

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

 # 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:
P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17
 36  46  70  52  44  61  42  40  35  53  50  67  48  56  42  54  43

 # Percentage of missing data:
[1] 2.410533

 # Observed heterozygosity:
       L1        L2        L3        L4        L5        L6        L7        L8
0.6118143 0.6666667 0.6793249 0.6455696 0.6329114 0.5654008 0.6497890 0.5949367
       L9
0.4514768

 # Expected heterozygosity:
       L1        L2        L3        L4        L5        L6        L7        L8
0.8657224 0.7928751 0.7953319 0.7603095 0.8702576 0.6884669 0.8157881 0.7603493
       L9
0.6062686
```

```
> names(toto)
```

```
[1] "N"        "pop.eff"  "loc.nall" "pop.nall" "NA.perc"  "Hobs"     "Hexp"
```

```
> par(mfrow = c(2, 2))
> plot(toto$pop.eff, toto$pop.nall, xlab = "Colonies sample size",
+     ylab = "Number of alleles", main = "Alleles numbers and sample sizes",
+     type = "n")
> text(toto$pop.eff, toto$pop.nall, lab = names(toto$pop.eff))
> barplot(toto$loc.nall, ylab = "Number of alleles", main = "Number of alleles per locus")
> barplot(toto$Hexp - toto$Hobs, main = "Heterozygosity: expected-observed",
+     ylab = "Hexp - Hobs")
> barplot(toto$pop.eff, main = "Sample sizes per population", ylab = "Number of genotypes",
+     las = 3)
```



Is mean observed H significantly lower than mean expected H ?

```
> bartlett.test(list(toto$Hexp, toto$Hobs))

        Bartlett test of homogeneity of variances

data:  list(toto$Hexp, toto$Hobs)
Bartlett's K-squared = 0.2556, df = 1, p-value = 0.6132


> t.test(toto$Hexp, toto$Hobs, pair = T, var.equal = TRUE, alter = "greater")

        Paired t-test

data:  toto$Hexp and toto$Hobs
t = 9.4044, df = 8, p-value = 6.7e-06
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1299211       Inf
sample estimates:
mean of the differences
              0.1619421
```

Yes.

3.3 Testing for structuration among populations

The G-statistic test (Goudet *et al.*, 1996) is implemented for `genind` objects and produces a `randtest` object (package ade4). The function to use is `gstat.randtest`, and requires the package *hierfstat.*:

```
> library(ade4)
> toto <- gstat.randtest(nancycats, nsim = 99)
> toto


Monte-Carlo test
Call: gstat.randtest(x = nancycats, nsim = 99)

Observation: 3416.974

Based on 99 replicates
Simulated p-value: 0.01
Alternative hypothesis: greater

    Std.Obs Expectation    Variance
   30.28628  1765.47259  2973.48636


> plot(toto)
```

Now that the test is performed, one can ask for F statistics. To get these, data are first converted to be used in the hierfstat package:

```
> library(hierfstat)
> toto <- genind2hierfstat(nancycats)
> head(toto)


    pop   fca8  fca23  fca43  fca45  fca77  fca78  fca90  fca96  fca37
001   1     NA 136146 139139 116120 156156 142148 199199 113113 208208
002   1     NA 146146 139145 120126 156156 142148 185199 113113 208208
003   1 135143 136146 141141 116116 152156 142142 197197 113113 210210
004   1 133135 138138 139141 116126 150150 142148 199199  91105 208208
005   1 133135 140146 141145 126126 152152 142148 193199 113113 208208
006   1 135143 136146 145149 120126 150156 148148 193195  91113 208208


> varcomp.glob(toto$pop, toto[, -1])


$loc
                1            2         3
fca8  0.08867161  0.116693199 0.6682028
fca23 0.05384247  0.077539920 0.6666667
fca43 0.05518935  0.066055996 0.6793249
fca45 0.05861271 -0.001026783 0.7083333
fca77 0.08810966  0.156863586 0.6329114
fca78 0.04869695  0.079006911 0.5654008
fca90 0.07540329  0.097194716 0.6497890
fca96 0.07538325 -0.005902071 0.7543860
fca37 0.04264094  0.116318729 0.4514768

$overall
```

```
      Pop       Ind     Error
0.5865502 0.7027442 5.7764917

$F
              Pop       Ind
Total 0.08301274 0.1824701
Pop   0.00000000 0.1084610
```

F statistics are provided in \$F; for instance, here, F_{st} is 0.083.

3.4 Testing for Hardy-Weinberg equilibrium

The Hardy-Weinberg equilibrium 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`).

```
> toto <- HWE.test.genind(nancycats, res = "matrix")
> dim(toto)
```

```
[1] 17  9
```

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(toto)
```

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

```
> which(toto < 1e-04, TRUE)
```

```
    row col
P06   6   2
P10  10   7
P10  10   8
P13  13   9
```

Here only 4 tests indicate departure from HW. Rows give populations, columns give markers. Now complete tests are returned, but the significant ones are already known.

```
> toto <- HWE.test.genind(nancycats, res = "full")
> toto$fca23$P06
```

```
        Pearson's Chi-squared test

data:  tab
X-squared = 49.7996, df = 15, p-value = 1.298e-05
```

```
> toto$fca90$P10


        Pearson's Chi-squared test

data:  tab
X-squared = 56.7523, df = 15, p-value = 9.04e-07


> toto$fca96$P10


        Pearson's Chi-squared test

data:  tab
X-squared = 92.0716, df = 15, p-value = 4.067e-13


> toto$fca37$P13


        Pearson's Chi-squared test

data:  tab
X-squared = 30.0206, df = 6, p-value = 3.896e-05
```

3.5 Performing a Principal Component Analysis on `genind` objects

The tables contained in `genind` objects can be submitted to a Principal Component Analysis (PCA) to seek a typology of individuals. Such analysis is straightforward using *adegenet* to prepare data and *ade4* for the analysis *per se*. One has first to replace missing data. Putting each missing observation at the mean of the concerned allele frequency seems the best choice (NA will be stuck at the origin).

```
> data(microbov)
> any(is.na(microbov$tab))


[1] TRUE
```

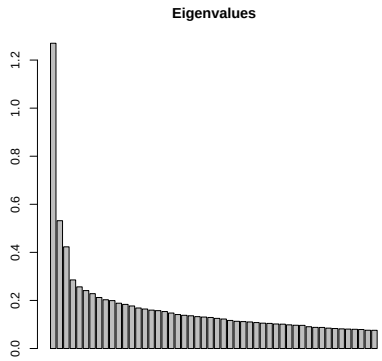
There are missing data. Assuming that these are evenly distributed (for illustration purpose only!), we replace them. Note that during conversion to `genind`, automatic replacement for missing data is available. This is not the case here as data are already imported and original file is not available.

```
> f1 <- function(v) {
+     v[is.na(v)] <- mean(v, na.rm = TRUE)
+     return(v)
+ }
> microbov$tab <- apply(microbov$tab, 2, f1)
> any(is.na(microbov$tab))
```

```
[1] FALSE
```

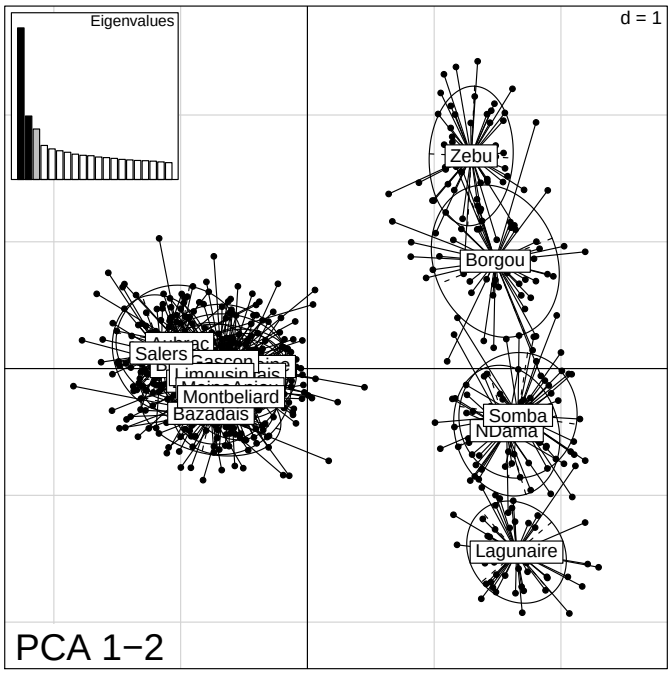
Well. Now, the analysis can be performed. Data are centred but not scaled as units are the same.

```
> pca1 <- dudi.pca(microbov$tab, cent = TRUE, scale = FALSE, scannf = FALSE,
+     nf = 3)
> barplot(pca1$eig[1:50], main = "Eigenvalues")
```



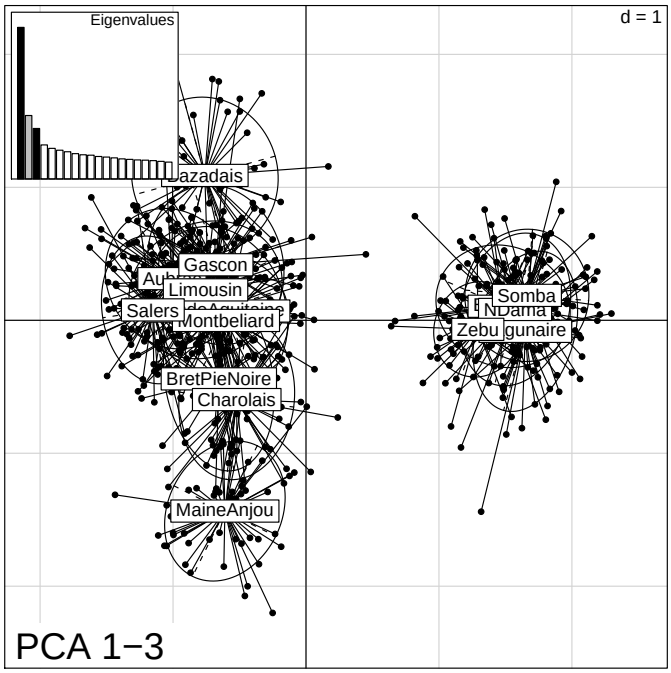
Here we represent the genotypes and 95% inertia ellipses for populations.

```
> s.class(pca1$li, microbov$pop, lab = microbov$pop.names, sub = "PCA 1-2",
+     csub = 2)
> add.scatter.eig(pca1$eig[1:20], nf = 3, xax = 1, yax = 2, posi = "top")
```



This plane shows that the main structuring is between African an French breeds, the second structure reflecting genetic diversity among African breeds. The third axis reflects the diversity among French breeds: Overall, all breeds seem well differentiated.

```
> s.class(pca1$li, microbov$pop, xax = 1, yax = 3, lab = microbov$pop.names,
+     sub = "PCA 1-3", csub = 2)
> add.scatter.eig(pca1$eig[1:20], nf = 3, xax = 1, yax = 3, posi = "top")
```

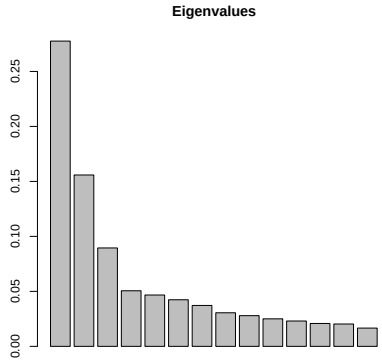


3.6 Performing a Correspondance Analysis on genpop objects

Being contingency tables, the tables contained in `genpop` objects can be submitted to a Correspondance Analysis (CA) to seek a typology of populations. The approach is very similar to the previous one for PCA. Missing data are first replaced during convertion from `genind`.

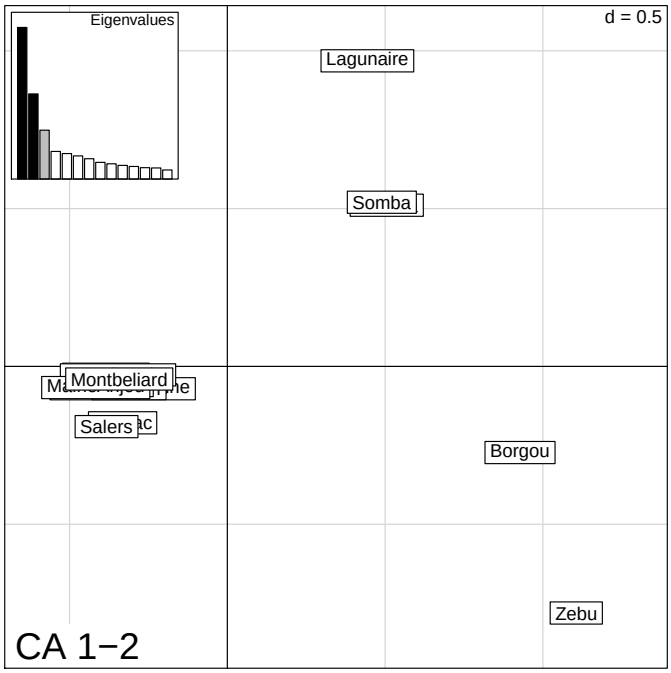
```
> data(microbov)
> toto <- genind2genpop(microbov, missing = "replace")


 Converting data from a genind to a genpop object...

...done.
```

```
> ca1 <- dudi.coa(as.data.frame(toto$tab), scannf = FALSE, nf = 3)
> barplot(ca1$eig, main = "Eigenvalues")
```

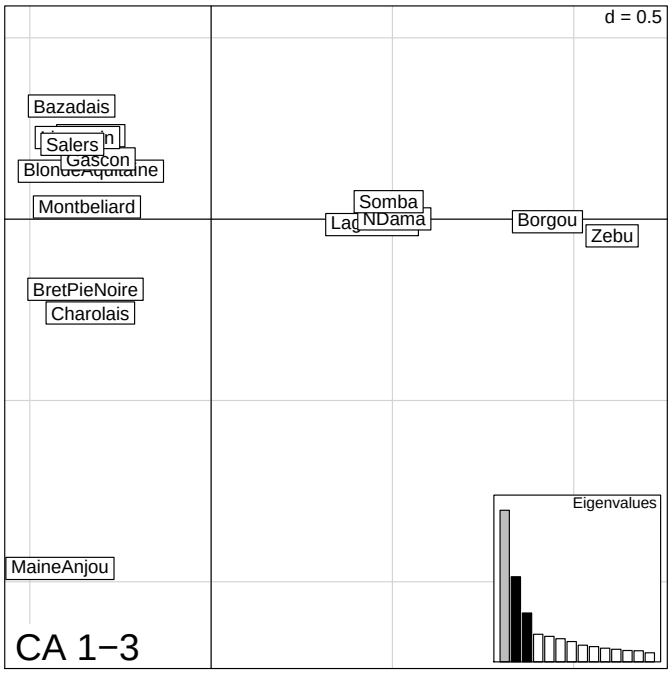


Now we display the resulting typologies:

```
> s.label(ca1$li, lab = toto$pop.names, sub = "CA 1-2", csub = 2)
> add.scatter.eig(ca1$eig, nf = 3, xax = 1, yax = 2, posi = "top")
```




```
> s.label(ca1$li, xax = 1, yax = 3, lab = toto$pop.names, sub = "CA 1-3",
+     csub = 2)
> add.scatter.eig(ca1$eig, nf = 3, xax = 2, yax = 3, posi = "bottomright")
```



Once again, axes are to be interpreted separately in terms of continental differentiation, a among-breed diversities.

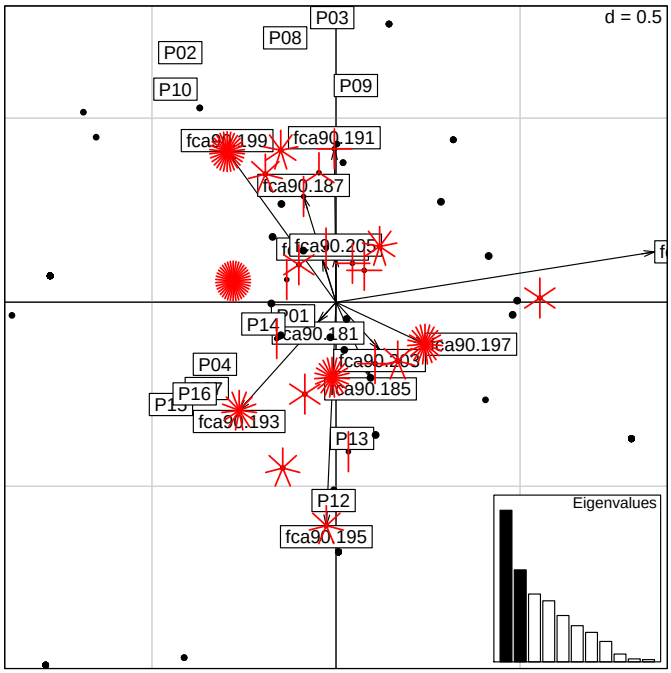
3.7 Analyzing a single locus

Here the emphasis is put on analyzing a single locus using different methods. Any marker can be isolated using the `seploc` instruction.

```
> data(nancycats)
> toto <- seploc(nancycats, truenames = TRUE)
> X <- toto$fca90
```

`fca90.ind` is a table containing only genotypes for the marker fca90. It can be analyzed, for instance, using an inter-class PCA. This analyzis provides a typology of individuals having maximal inter-colonies variance.

```
> library(ade4)
> pcaX <- dudi.pca(X, cent = T, scale = F, scannf = FALSE)
> pcabetX <- between(pcaX, nancycats$pop, scannf = FALSE)
> s.arrow(pcabetX$c1, xlim = c(-0.9, 0.9))
> s.class(pcabetX$ls, nancycats$pop, cell = 0, cstar = 0, add.p = T)
> sunflowerplot(X %*% as.matrix(pcabetX$c1), add = T)
> add.scatter.eig(pcabetX$eig, xax = 1, yax = 2, posi = "bottomright")
```



Here the differences between individuals are mainly expressed by three alleles: 199, 197 and 193. However, there is no clear structuration to be seen at an individual level. Is F_{st} significant taking only this marker into account? We perform the G-statistic test and enventually compute the corresponding F statistics.

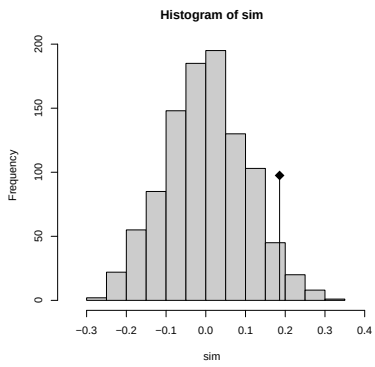
```
> fca90.ind <- as.genind(X, pop = nancycats$pop)
> gstat.randtest(fca90.ind, nsim = 999)

Monte-Carlo test
Call: gstat.randtest(x = fca90.ind, nsim = 999)

Observation: 437.135

Based on 999 replicates
Simulated p-value: 0.001
Alternative hypothesis: greater

    Std.Obs Expectation    Variance
   14.53824   188.45715   292.58397

> F <- varcomp(genind2hierfstat(fca90.ind))$F
> rownames(F) <- c("tot", "pop")
> colnames(F) <- c("pop", "ind")
> F

           pop       ind
tot 0.09168833 0.2098744
pop 0.00000000 0.1301162
```

In this case the information is best summarized by F statistics than by an ordination method. It is likely because all colonies are differentiated but none forming groups of related colonies.

3.8 Testing for isolation by distance

Isolation by distance (IBD) is tested using Mantel test between a matrix of genetic distances and a matrix of geographic distances. It can be tested using individuals as well as populations. This example uses cat colonies. We use Edwards' distance *versus* Euclidean distances between colonies.

```
> data(nancycats)
> toto <- genind2genpop(nancycats, miss = "0")


 Converting data from a genind to a genpop object...

...done.


> Dgen <- dist.genpop(toto, method = 2)
> Dgeo <- dist(nancycats$xy)
> library(ade4)
> ibd <- mantel.randtest(Dgen, Dgeo)
> ibd


Monte-Carlo test
Call: mantel.randtest(m1 = Dgen, m2 = Dgeo)

Observation: 0.1858000

Based on 999 replicates
Simulated p-value: 0.039
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
 1.773585307 -0.003021817  0.011334448


> plot(ibd)
```



Isolation by distance is indeed significant.

References

Chessel, D., Dufour, A.-B. & Thioulouse, J. (2004). The ade4 package-I- one-table methods. *R News* **4**, 5–10.

Goudet, J. (2005). Hierfstat, a package for r to compute and test hierarchical f-statistics. *Molecular Ecology Notes* **5**, 184–186.

Goudet, J., Raymond, M., Meeüs, T. & Rousset, F. (1996). Testing differentiation in diploid populations. *Genetics* **144**, 1933–1940.

Ihaka, R. & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* **5**, 299–314.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org`. ISBN 3-900051-07-0.