Genetic clustering and hybrid detection using *adegenet*

Thibaut Jombart *

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

November 14, 2016

Abstract

This tutorial presents an overview of likelihood-based genetic clustering in *adegenet*, as implemented by the function `genclust.em`. After a brief presentation of the rationale of the method, we illustrate its use in two situations: for identifying genetic clusters, and then for detecting hybrids.

*`thibautjombart@gmail.com`

Contents

1 The method, in a nutshell

1.1 Model formulation

1.1.1 Notations

The following notations will be used to describe the model.

- $i = 1, \ldots, I$: index of individuals
- $j = 1, \ldots, J$: index of loci
- $k = 1, \ldots, K$: index of groups
- $x_{i,j}$: vector of allele counts for individual i at locus j
- x_i: vector of allele counts of individual i for all loci, i.e. concatenation $[x_{i,1} \ldots x_{i,J}]$
- x: matrix of allele counts, obtained by row-concatenation of the x_i
- g_i: group of individual i ($g_i \in \{1, \ldots, K\} \ \forall \ i \in \{1, \ldots, I\}$)
- g: vector of group memberships ($g = g_1, \ldots, g_I$)
- $f_{k,j}$: vector of alle frequencies of group k at locus j
- f_k: vector of alle frequencies of group k for all loci, i.e. concatenation $[f_{k,1} \ldots x_{k,J}]$
- x: matrix of all allele frequencies, obtained by row-concatenation of the f_k
- π: the ploidy of all individuals

1.1.2 General likelihood

Likelihood-based genetic clustering in *adegenet* is implemented by the function `genclust.em`. This approach uses Hardy-Weinberg's equilibrium to define the probabilities of observing given genotypes under known allele frequencies. For now, we assume the group to which i belongs is known (noted g_i), and has known allele frequencies $f_{g_i,j}$. For any level of ploidy π and any codominant marker, the likelihood of $x_{i,j}$ is defined as:

$$p(x_{i,j}|f_{g_i,j}, \pi) = \mathcal{M}(x_{i,j}, f_{g_i,j}, \pi) \tag{1}$$

where $\mathcal{M}$ is the multinomial probability mass function.

Assuming independence between loci ($j = 1, \ldots, J$), the likelihood of individual i across all loci is:

$$p(x_i|f_{g_i}, \pi) = \prod_j p(x_{i,j}|f_{g_i,j}, \pi) \tag{2}$$

Assuming further independence between individuals, conditional on their group membership g and all allele frequencies f, the total likelihood is defined as:

$$p(x|f, g, \pi) = \prod_i p(x_i|f_{g_i}, \pi) \tag{3}$$

So that the general equation for the total log-likelihood of the model is computed as:

$$\mathcal{LL}(x, f, g, \pi) = \sum_i \sum_j \log(\mathcal{M}(x_{i,j}, f_{g_i,j}, \pi)) \tag{4}$$

Note that this model could easily accomodate varying ploidy through π_i, π_j or $\pi_{i,j}$ but we leave this out for now.

1.1.3 Useful particular cases

As particular case, the likelihood of all possible diploid genotypes with alleles A and B is defined, noting f_{g_i} the vector of allele frequencies in a given group g_i as:

$$p(AA|f_{g_i}, 2) = f_A^2 \tag{5}$$
$$p(B|f_{g_i}, 2) = f_B^2 \tag{6}$$
$$p(AB|f_{g_i}, 2) = 2 f_A f_B \tag{7}$$

Note that for haploid data, this is even simpler:

$$p(A|f_{g_i}, 1) = f_A \tag{8}$$
$$p(B|f_{g_i}, 1) = f_B \tag{9}$$

1.1.4 Group membership probabilities

Assuming any individual i comes from one of the sampled groups $k = 1, \ldots, K$, the probability that i belongs to group k is defined as the standardized likelihood:

$$p(g_i = k) = \frac{p(x_i|g_i = k, f_k, \pi)}{\sum_q p(x_i|g_i = q, f_q, \pi)} \tag{10}$$

which ensures that:

$$\sum_k p(g_i = k) = 1 \tag{11}$$

1.2 Estimation using the EM algorithm

The model formulation above supposes that both the groups g, and the allele frequencies in these groups f, are known. In practice, though, these need to be estimated. This is achieved using the expectation-maximization algorithm, which we apply as follow:

1. define initial groups for invididuals, g
2. (*expectation*) compute allele frequencies f and then $p(g_i = k)$ for all i and k
3. (*maximization*) assign individuals to their most likely group
4. return to 2) until convergence

Here, we consider that the algorithm has converged when the change in the global log-likelihood is less than 1e-14. The advantage of this algorithm is that it converges very fast, typically in less than 10 iterations. The first step can be achieved using random group allocation, in which case several runs of the EM algorithm can be useful to ensure that the best solution (with highest log-likelihood) is attained. Alternatively, clusters can be defined by another fast clustering method, such as the k-means implemented in `find.clusters`. This latter option is used by default in our implementation.

1.3 Identifying hybrids

The allele frequency f_{h_w} in a hybrid population h_w is modelled as weighted averages of the allele frequencies in the parental populations k and q:

$$f_{h_w} = w f_k + (1 - w) f_q \tag{12}$$

where w has value between 0 and 1. Typical values of w are 0.5 for F1 hybrids, 0.75 for backcrosses F1/1, 0.25 for backcrosses F1/2, etc.

The EM algorithm described above can be extended to account for various hybrids using:

1. define initial groups for invididuals, g
2. (expectation) define parental populations as the two groups most distant from each other; compute allele frequencies for parental populations f_k and f_q and subsequently for all hybrid populations h_w, and then $p(g_i = k)$, $p(g_i = q)$ and $p(g_i = h_w)$ for all i and h_w
3. (maximization) assign individuals to their most likely group
4. return to 2) until convergence

2 Example using simulated data

2.1 In the absence of hybrids: DAPC data revisited

The method described above is implemented in the function `genclust.em`. To illustrate it, we first re-analyse datasets originally simulated to illustrate the DAPC (Jombart *et al.* 2010, BCM Genetics). The data `dapcIllus` contains for datasets:

- `$a`: island model with 6 demes
- `$b`: hierarchical island model with 6 demes (3,2,1)
- `$c`: one-dimensional stepping stone model with 12 demes
- `$d`: one-dimensional stepping stone with a boundary in between sets of 12 demes

```
library(adegenet)
data(dapcIllus)
sapply(dapcIllus, nPop)

##  a  b  c  d
##  6  6 12 24
```

For each dataset, we identify clusters using `genclust.em`, indicating the right number of clusters, and compare the obtained clusters to the true clusters.

```
a.clust <- genclust.em(dapcIllus$a, k = 6)
class(a.clust)

## [1] "genclust.em" "list"

names(a.clust)

## [1] "group"     "ll"        "proba"     "converged" "n.iter"
```

The output of `genlcust.em` is a list containing the following information:

- `group`: a factor giving maximum-likelihood group assignment for each individual
- `ll`: the log-likelihood of the model
- `proba`: a matrix giving full group membership probabilities, with individuals in rows and clusters in column
- `converged`: a logical indicating if the algorithm converged before reaching the maximum number of iterations
- `n.iter`: the number of iterations after which the algorithm stopped

Let us examine these components:

```
head(a.clust$group, 12)

##  1  2  3  4  5  6  7  8  9 10 11 12
##  1  1  1  1  1  1  1  1  1  1  1  1
## Levels: 1 2 3 4 5 6

length(a.clust$group)

## [1] 600

a.clust$ll

## [1] -11933.4
```

```
dim(a.clust$proba)

## [1] 600   6

head(a.clust$proba)

##           1            2            3            4            5
## 1 0.9889863 1.967245e-05 2.750886e-06 1.099126e-02 1.864221e-09
## 2 1.0000000 2.744606e-12 2.685464e-12 3.936665e-10 4.165036e-10
## 3 1.0000000 3.925511e-10 1.029866e-09 1.018073e-08 1.846200e-13
## 4 0.9925573 2.253888e-05 7.419267e-03 8.688965e-07 9.585198e-10
## 5 1.0000000 1.007399e-11 4.500484e-08 1.473021e-10 1.049492e-09
## 6 0.9999987 3.898077e-11 1.266882e-06 2.842916e-11 2.134936e-09
##              6
## 1 3.080832e-10
## 2 1.644282e-15
## 3 1.872534e-11
## 4 1.500519e-09
## 5 9.216524e-11
## 6 3.928634e-11

head(round(a.clust$proba),3)

##   1 2 3 4 5 6
## 1 1 0 0 0 0 0
## 2 1 0 0 0 0 0
## 3 1 0 0 0 0 0

a.clust$converged

## [1] TRUE

a.clust$n.iter

## [1] 3
```

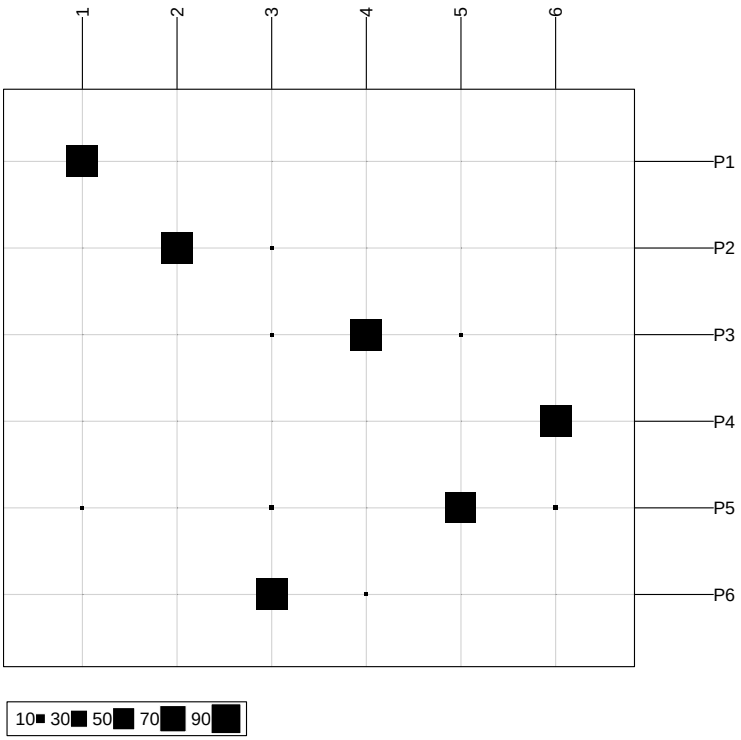
The inferred groups can be compared to the original ones using `table`, which builds a contingency table putting the original groups in rows, and the newly inferred groups in columns; results are visualised using `table.value`

```
a.tab <- table(pop(dapcIllus$a), a.clust$group)
a.tab

##
##        1   2   3   4   5   6
##   P1 100   0   0   0   0   0
```

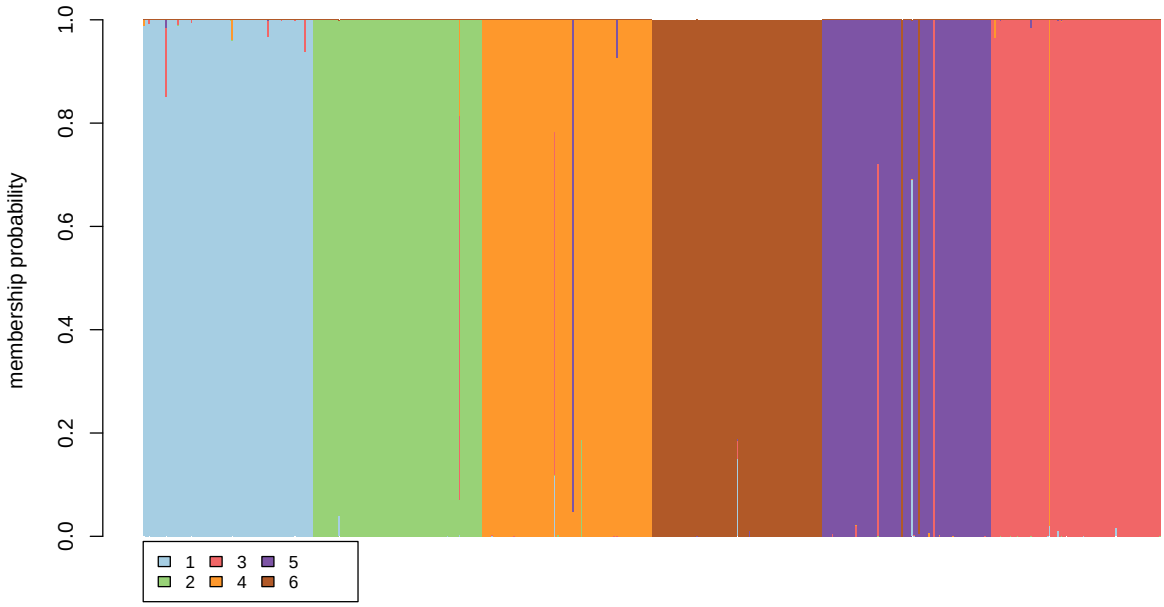
```
##    P2   0  99   1   0   0   0
##    P3   0   0   1  98   1   0
##    P4   0   0   0   0   0 100
##    P5   1   0   2   0  95   2
##    P6   0   0  99   1   0   0

table.value(a.tab, col.labels = 1:6)
```



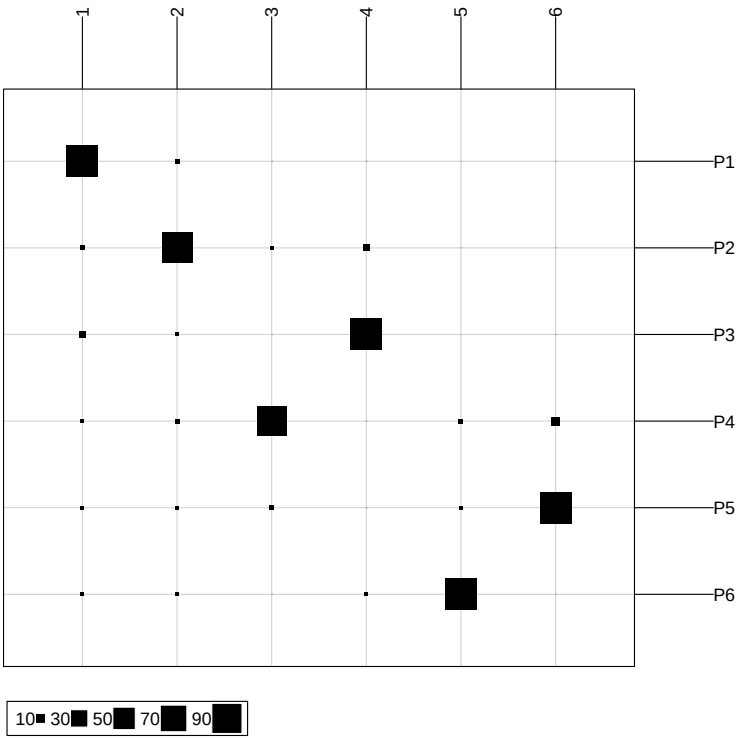
Here, original groups were nearly perfectly retrieved. Finally, we can visualise the group membership probabilities, which represent the probabilities that each genotype was generated under the various populations, using the `compoplot`:

```
compoplot(a.clust)
```

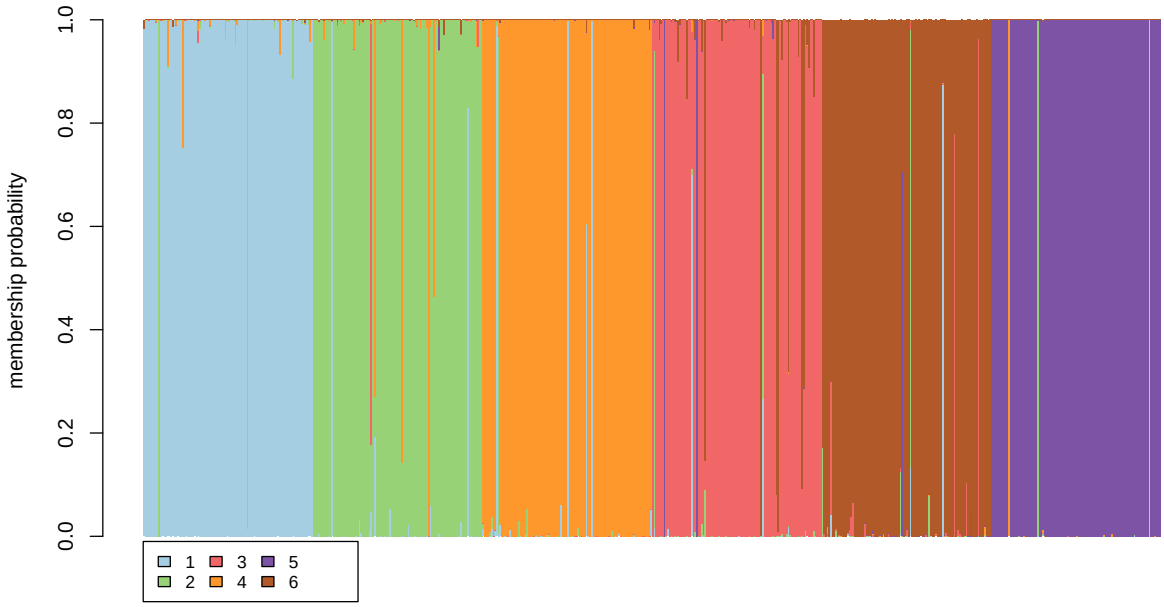


Here, assignment of individuals to their actual groups is essentially perfect; the few 'misclassified' individuals are effectively migrants. In fact, results remain very good in other models as well. Results are near perfect for the hierarchical island mode:

```
b.clust <- genclust.em(dapcIllus$b, k = 6)
b.tab <- table(pop(dapcIllus$b), b.clust$group)
table.value(b.tab, col.labels = 1:6)
```

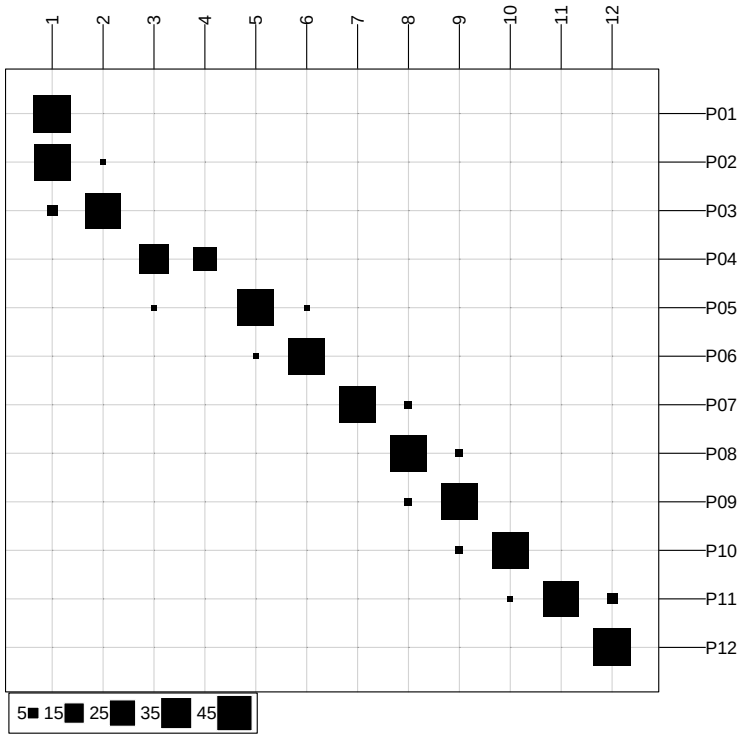



```
compoplot(b.clust)
```

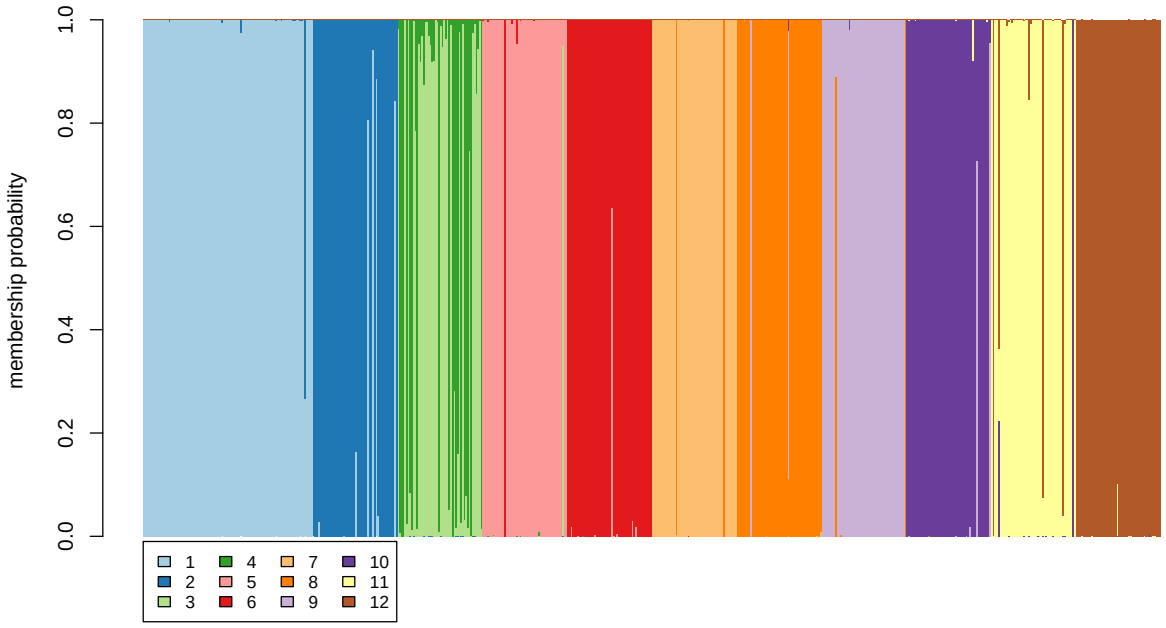


and for the stepping stone model:

```
c.clust <- genclust.em(dapcIllus$c, k = 12)
c.tab <- table(pop(dapcIllus$c), c.clust$group)
table.value(c.tab, col.labels = 1:12)
```

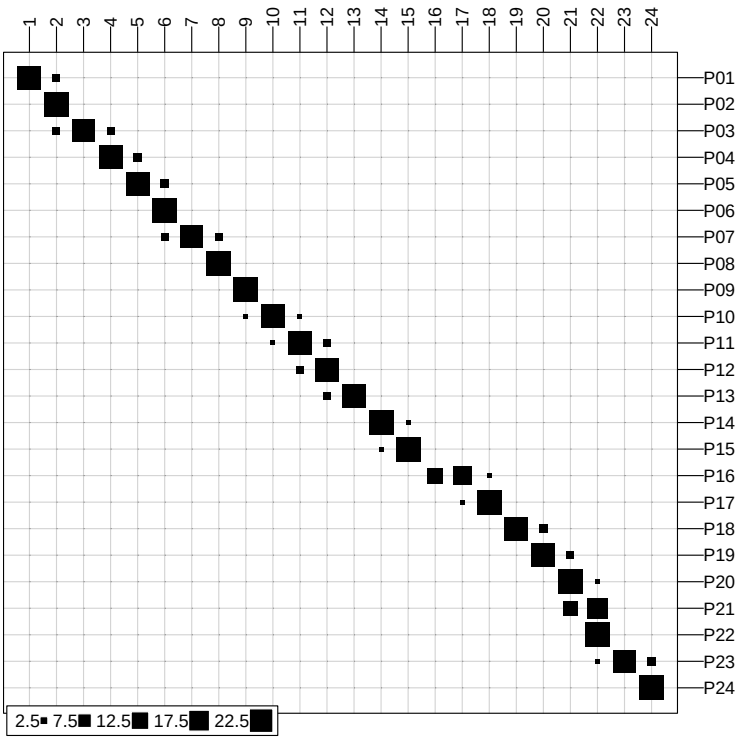



```
compoplot(c.clust)
```

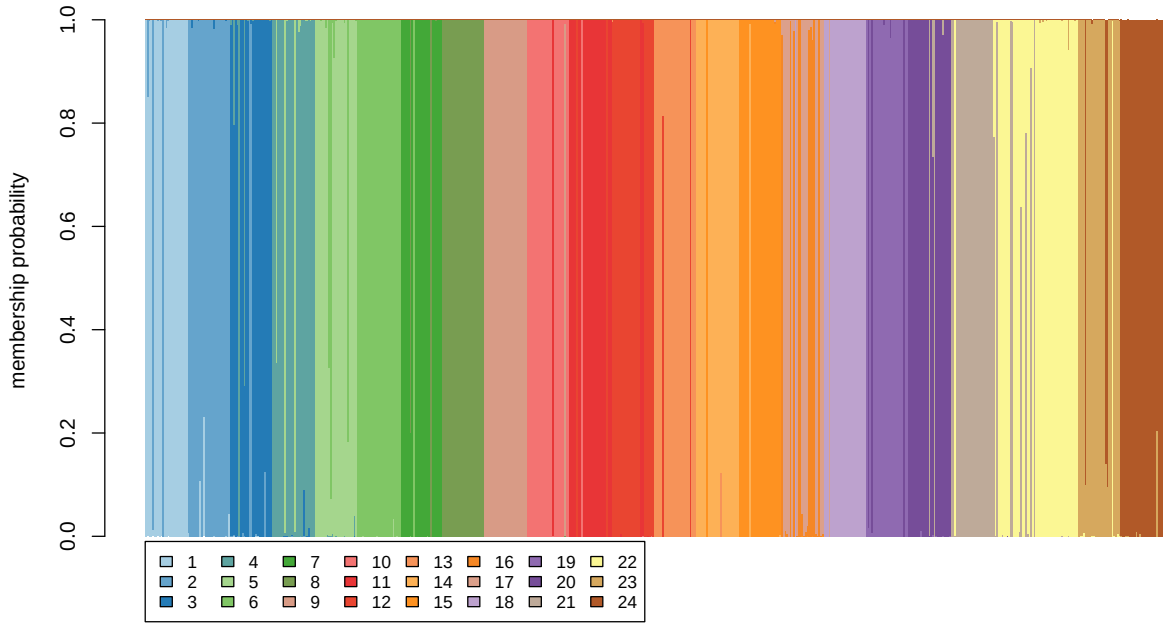


In the larger stepping stone model with a boundary, results remain very good but some neighbouring demes are harder to distinguish (especially 1-2, 3-4, 13-14 and 18-19).

```
d.clust <- genclust.em(dapcIllus$d, k = 24)
d.tab <- table(pop(dapcIllus$d), d.clust$group)
table.value(d.tab, col.labels = 1:24, csize = .6)
```

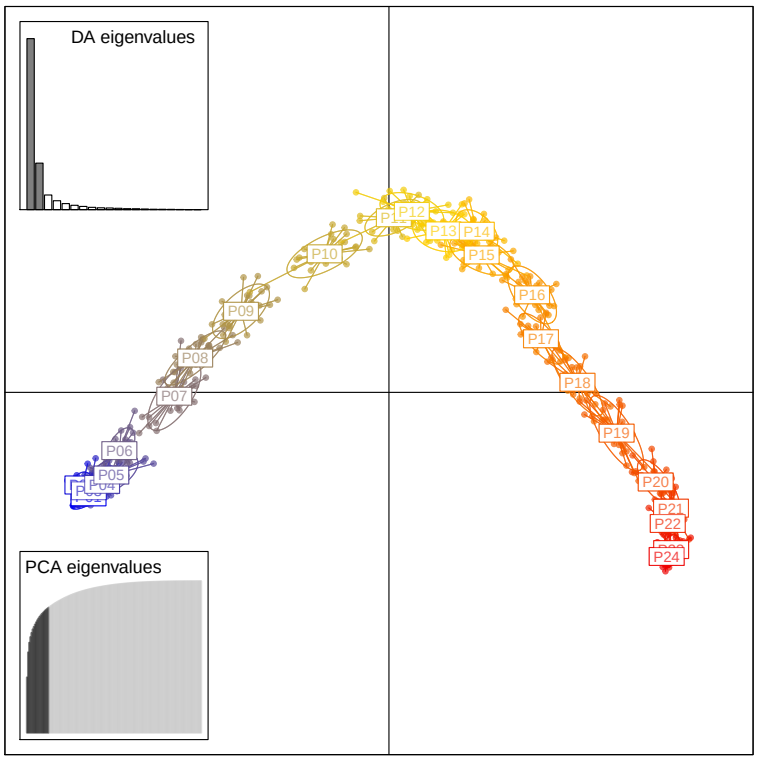



```
compoplot(d.clust, n.col=8)
```



However, note that this lack of distinction in some demes merely comes from a weak genetic differentiation:

```
d.dapc <- dapc(dapcIllus$d, n.pca = 20, n.da = 2)
scatter(d.dapc, clab = 0.85, col = deepseasun(24),
        posi.da="topleft", posi.pca = "bottomleft", scree.pca = TRUE)
```



2.2 Looking for hybrids

2.2.1 Simulating hybrids using `hybridize`

Simulate hybrids F1

```
set.seed(1)
data(microbov)

zebu <- microbov[pop="Zebu"]
salers <- microbov[pop="Salers"]
hyb <- hybridize(zebu, salers, n = 30)
x <- repool(zebu, salers, hyb)
```

Simulate hybrids backcross (F1 / parental)

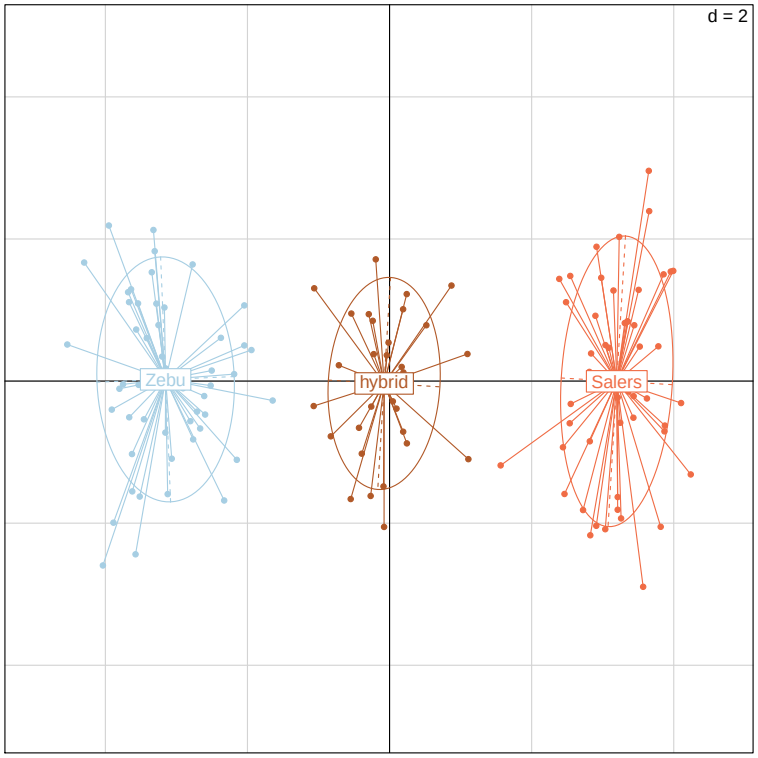
```
f1.zebu <- hybridize(hyb, zebu, 20, pop = "f1.zebu")
f1.salers <- hybridize(hyb, salers, 25, pop = "f1.salers")
y <- repool(x, f1.zebu, f1.salers)
```

2.2.2 Looking for F1 hybrids

`genclust.em` can also be used in the presence of hybridization. In this case, it will attempt to classify individuals into parental populations, or various types of hybrids defined by the user.

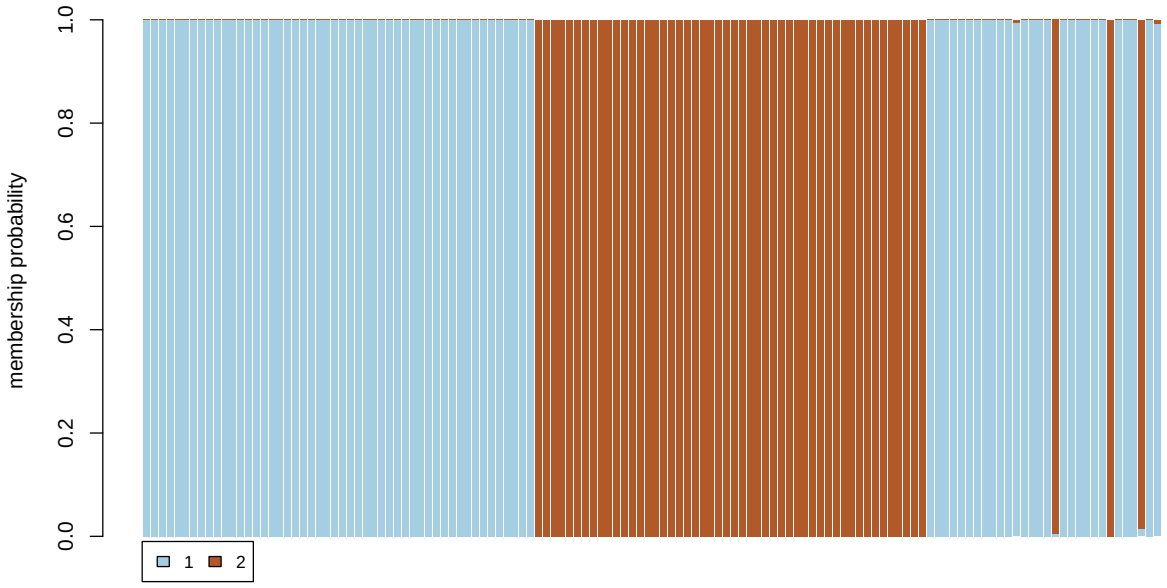
PCA:

```
x.pca <- dudi.pca(tab(x, NA.method = "mean"), scannf = FALSE, scale = FALSE)
s.class(x.pca$li, pop(x), col = funky(3))
```



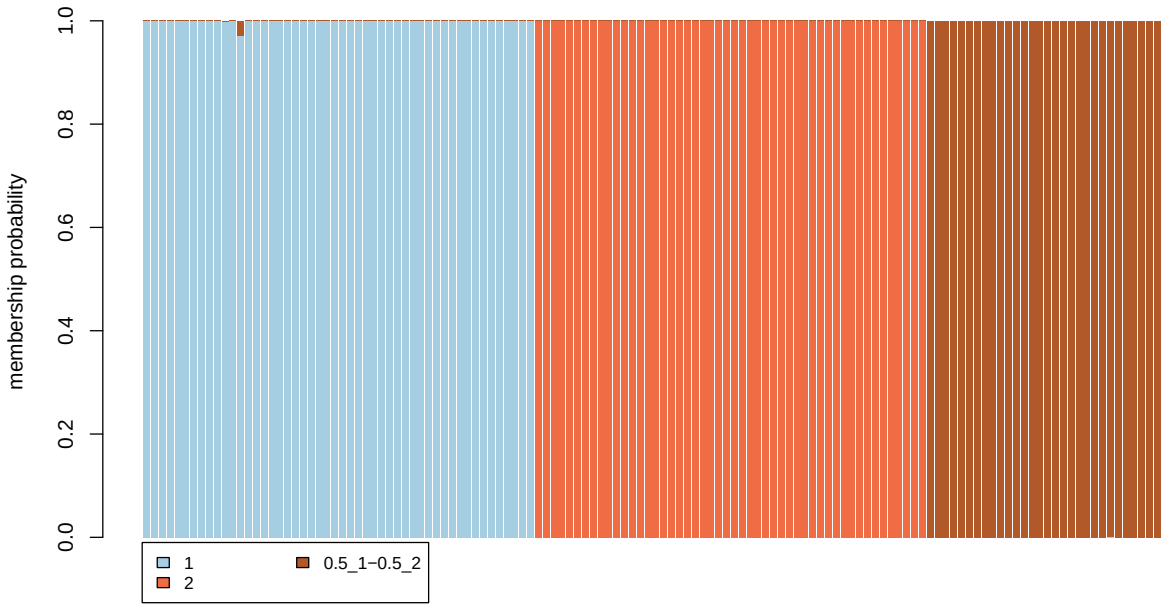
method without hybrids

```
res.no.hyb <- genclust.em(x, k = 2, hybrids = FALSE)
compoplot(res.no.hyb, n.col = 2)
```



method with hybrids

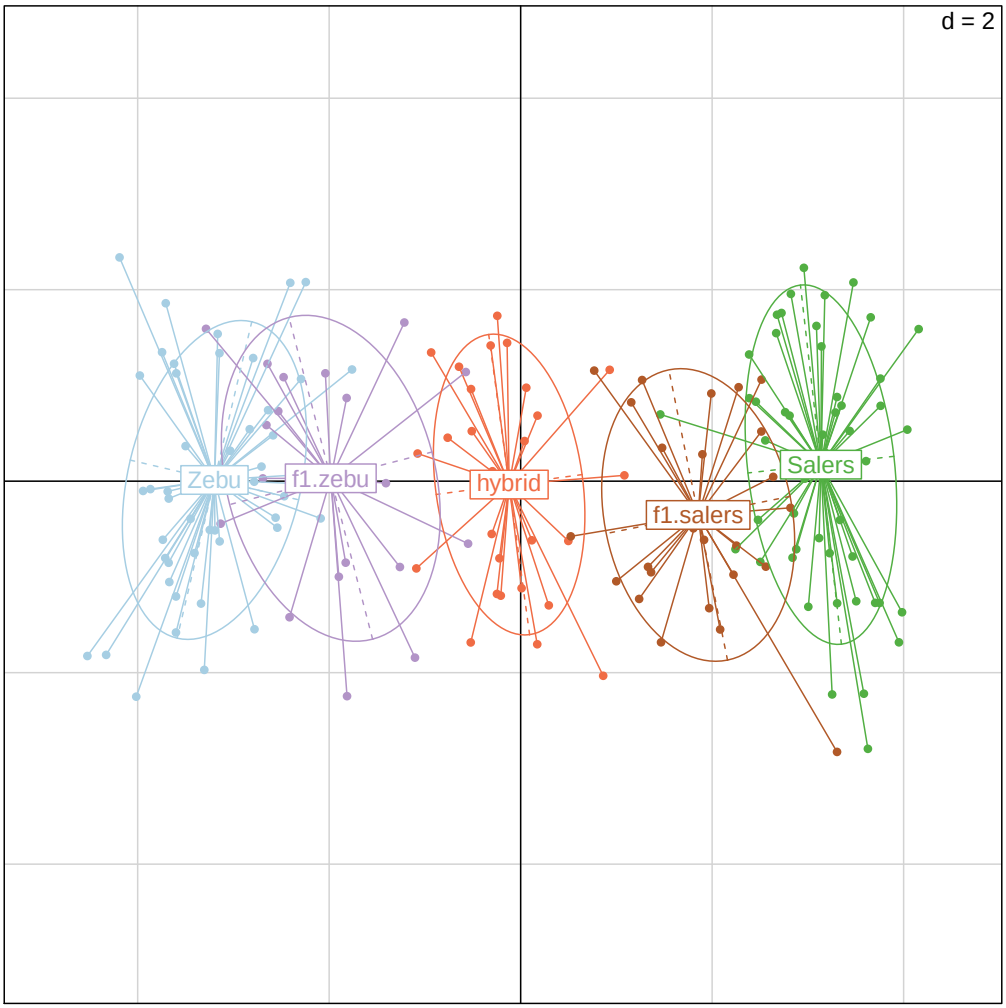
```
res.hyb <- genclust.em(x, k=2, hybrids=TRUE)
compoplot(res.hyb, n.col=2)
```



2.2.3 Looking for F1 and back-crosses

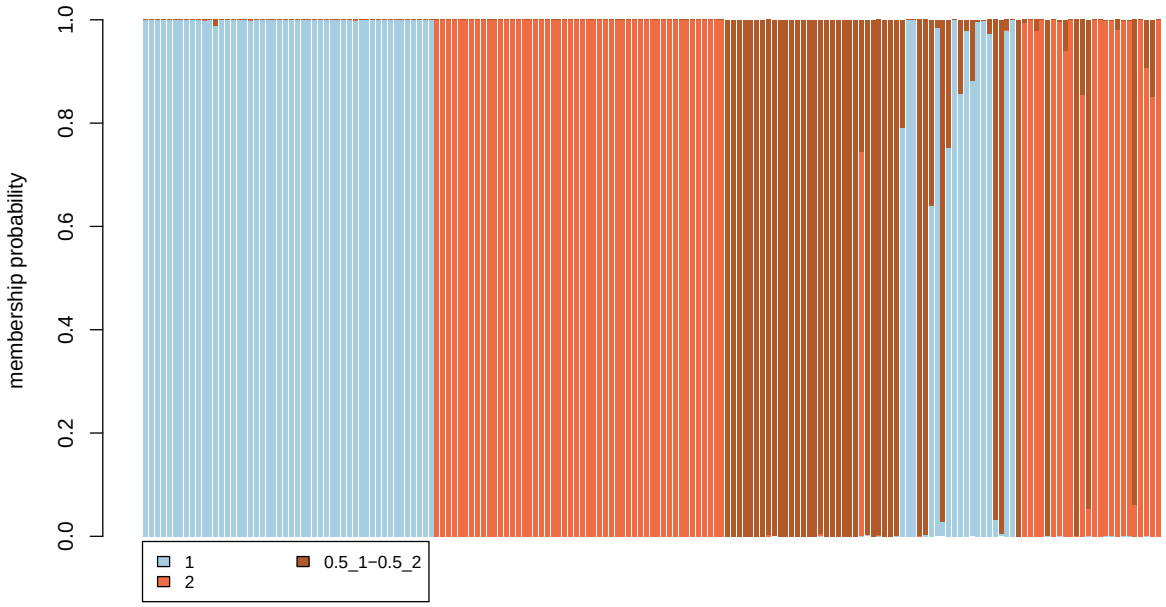
PCA:

```
y.pca <- dudi.pca(tab(y, NA.method = "mean"), scannf = FALSE, scale = FALSE)
s.class(y.pca$li, pop(y), col = funky(5))
```



method with hybrids F1 only

```
res2.hyb <- genclust.em(y, k = 2, hybrids = TRUE)
compoplot(res2.hyb, n.col=2)
```



method with back-cross

```
res2.back <- genclust.em(y, k=2, hybrids = TRUE, hybrid.coef = c(.25,.5))
compoplot(res2.back)
```