Genetic clustering and hybrid detection using *adegenet*

Thibaut Jombart *

Imperial College London

MRC Centre for Outbreak Analysis and Modelling

January 31, 2018

Abstract

This tutorial presents an overview of likelihood-based genetic clustering in *adegenet*, as implemented by the function `snapclust`. 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 `snapclust`. 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 `snapclust`. 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)

## Loading required package:  ade4
##
##    /// adegenet 2.1.1 is loaded ////////////
##
##    > overview:  '?adegenet'
##    > tutorials/doc/questions:  'adegenetWeb()'
##    > bug reports/feature requests:  adegenetIssues()

data(dapcIllus)
sapply(dapcIllus, nPop)

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

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

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

## [1] "snapclust" "list"

names(a.clust)

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

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
```

```r
length(a.clust$group)
```

```
## [1] 600
```

```r
a.clust$ll
```

```
## [1] -19536.21
```

```r
dim(a.clust$proba)
```

```
## [1] 600   6
```

```r
head(a.clust$proba)
```

```
##           1            2            3            4            5
## 1 0.9898886 1.806092e-05 2.525667e-06 1.009082e-02 2.475659e-09
## 2 1.0000000 2.852050e-12 2.789269e-12 4.088715e-10 7.755781e-10
## 3 1.0000000 3.607545e-10 9.469806e-10 9.360291e-09 4.026278e-13
## 4 0.9930270 2.111623e-05 6.951046e-03 8.133684e-07 1.567184e-09
## 5 1.0000000 1.686716e-11 4.304500e-08 1.349373e-10 1.686716e-09
## 6 0.9999988 4.747989e-11 1.208648e-06 4.747989e-11 3.228632e-09
##              6
## 1 4.951317e-10
## 2 2.989570e-15
## 3 1.731300e-11
## 4 1.567184e-09
## 5 8.433582e-11
## 6 4.747989e-11
```

```r
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
```

```r
a.clust$converged
```

```
## [1] TRUE
```

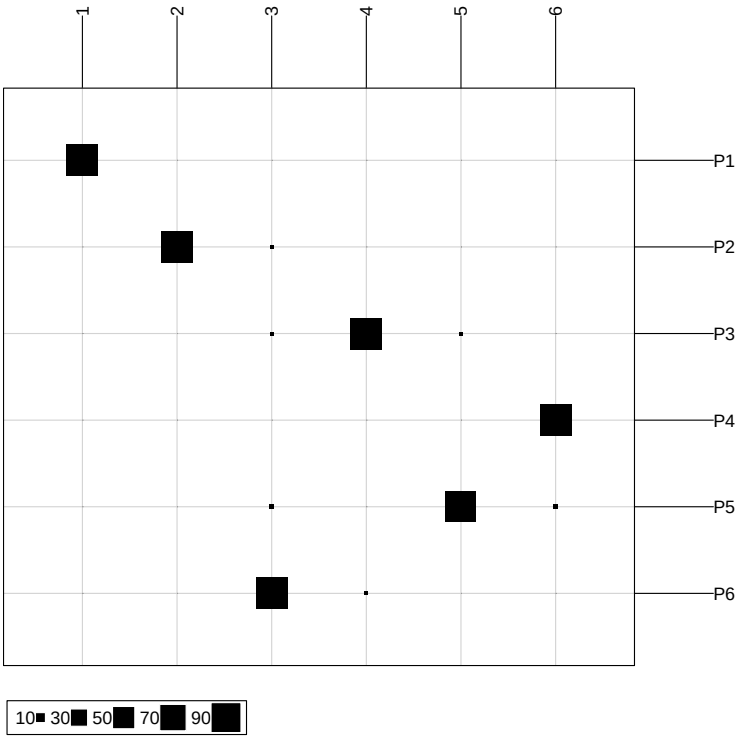
```r
a.clust$n.iter
```

```
## [1] 4
```

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

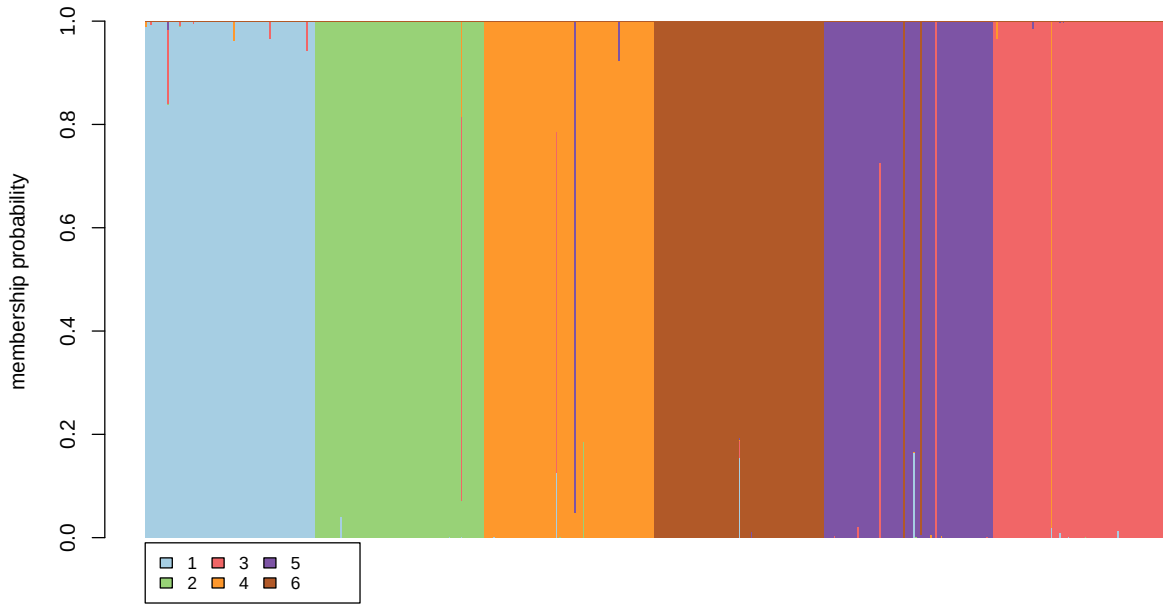
columns; results are visualised using `table.value`

```
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   0   0   2   0  96   2
##   P6   0   0  99   1   0   0

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



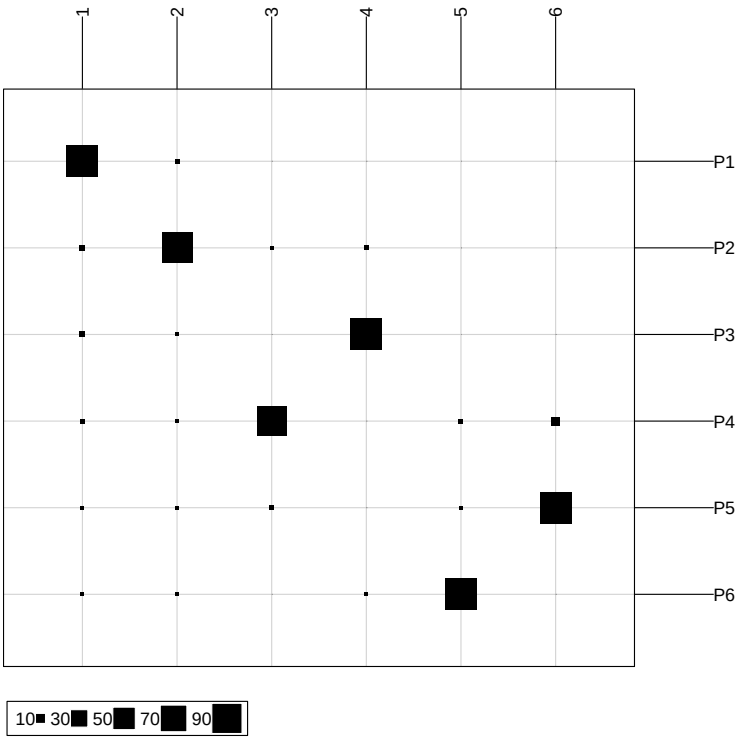
Here, original groups were nearly perfectly retrieved. Finally, we can visualise the group membership probabilities, which represent the probabilities that each genotype was generated under the various populations, using the `compoplot`:

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

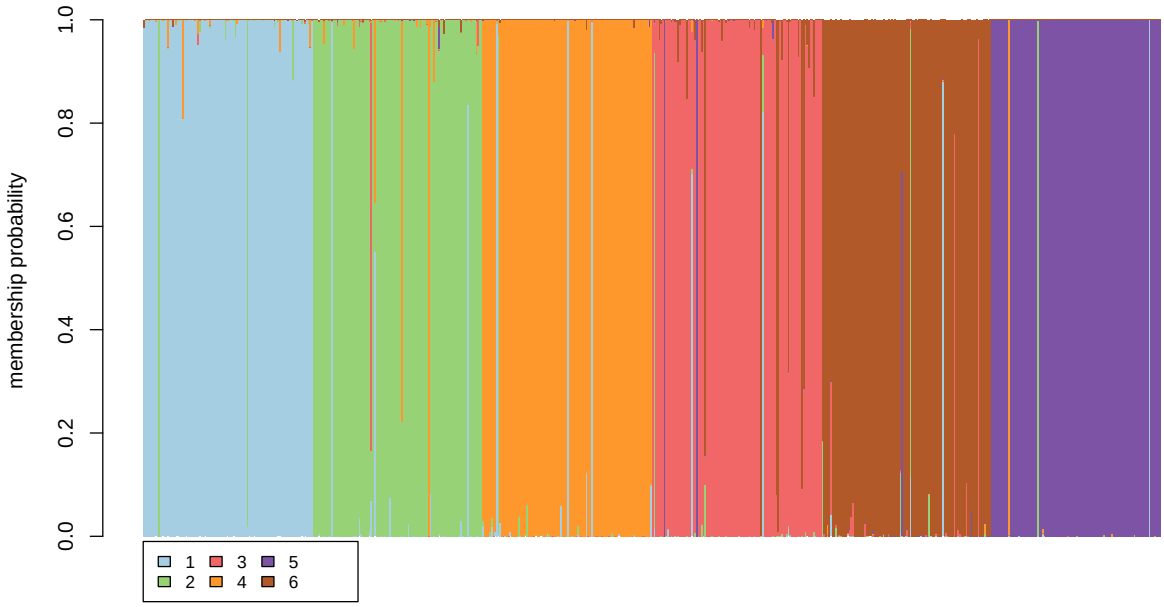


Here, assignment of individuals to their actual groups is essentially perfect; the few 'misclassified' individuals are effectively migrants. In fact, results remain very good in other models as well. Results are near perfect for the hierarchical island mode:

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

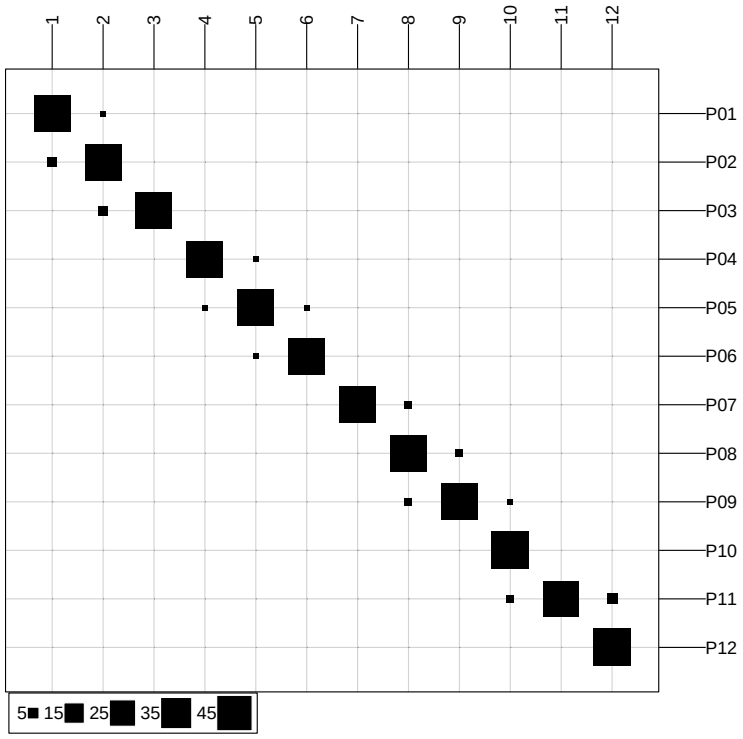



```
compoplot(b.clust)
```

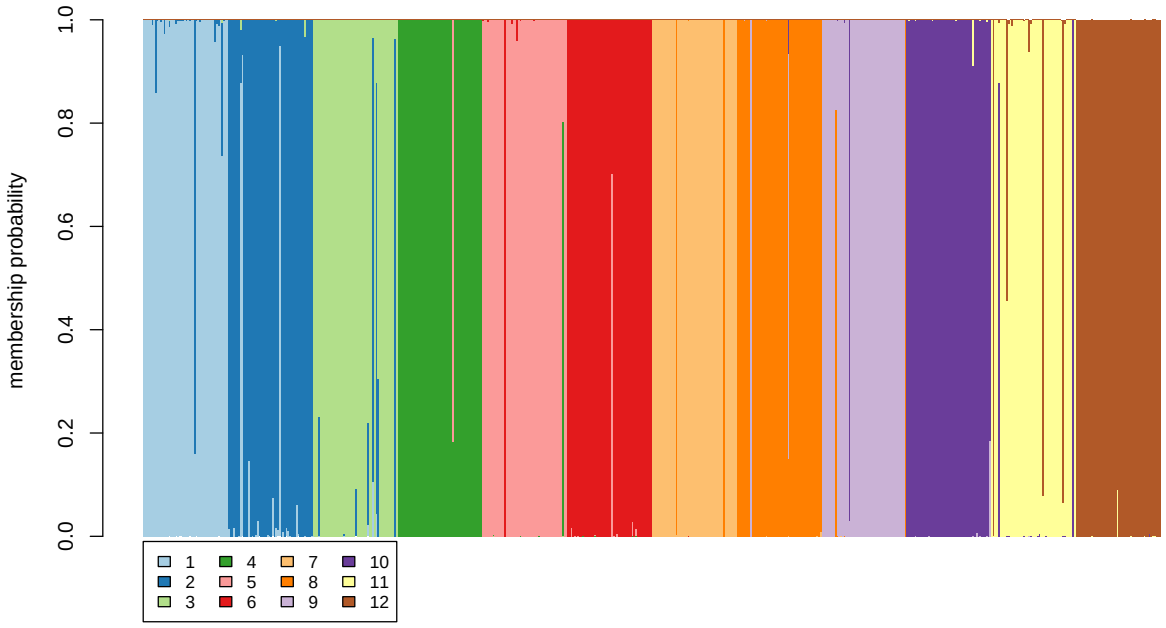


and for the stepping stone model:

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

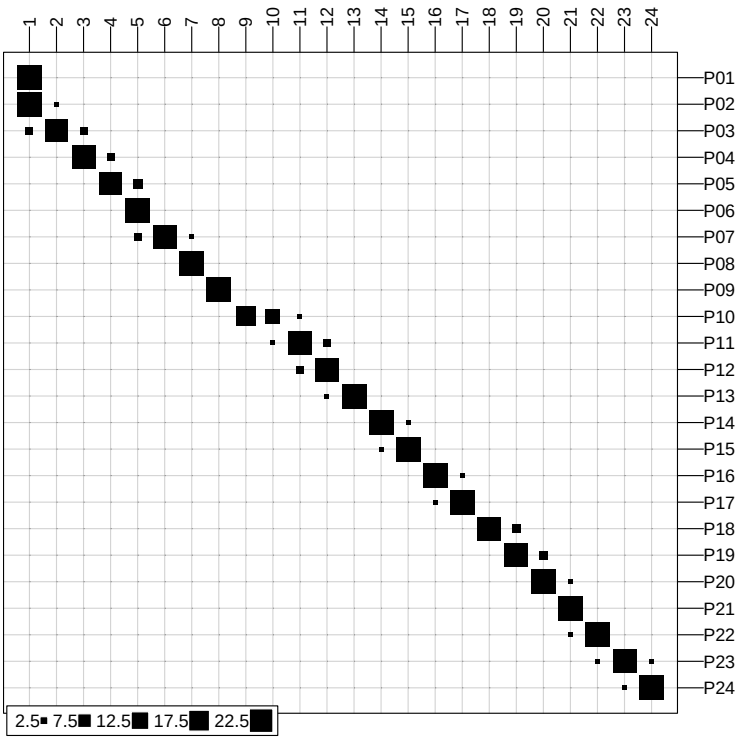



```
compoplot(c.clust)
```

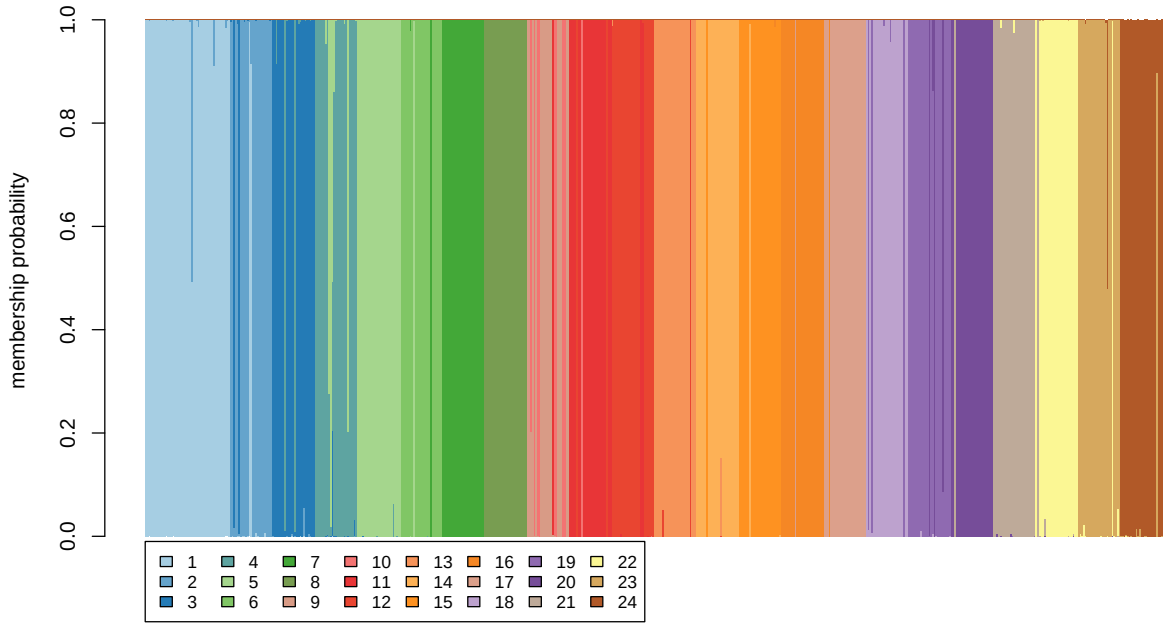


In the larger stepping stone model with a boundary, results remain very good but some neighbouring demes are harder to distinguish (especially 1-2, 3-4, 13-14 and 18-19).

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

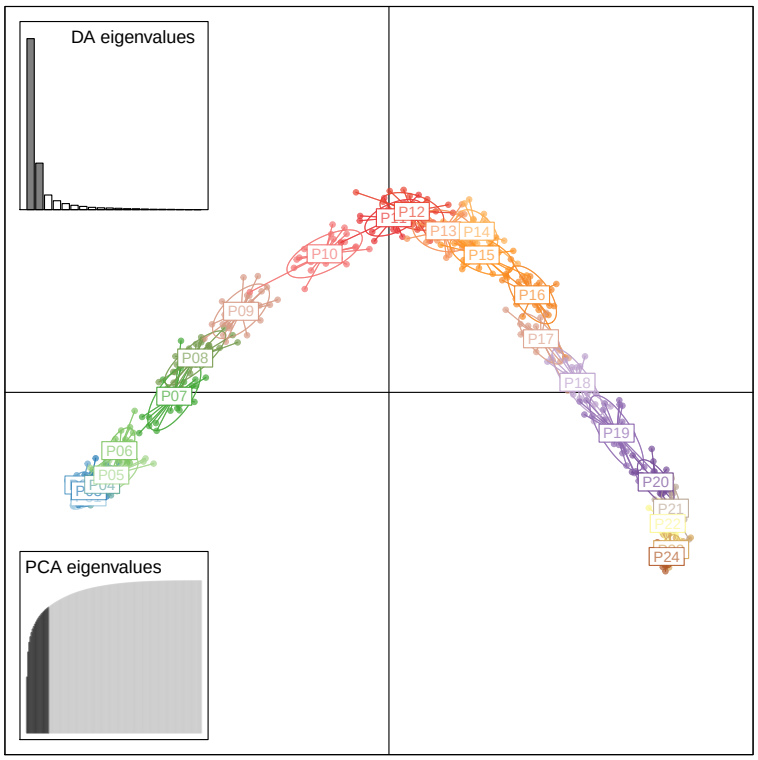



```
compoplot(d.clust, n.col=8)
```



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 = funky(24),
        posi.da="topleft", posi.pca = "bottomleft", scree.pca = TRUE)
```



2.2 Identifying 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)
```

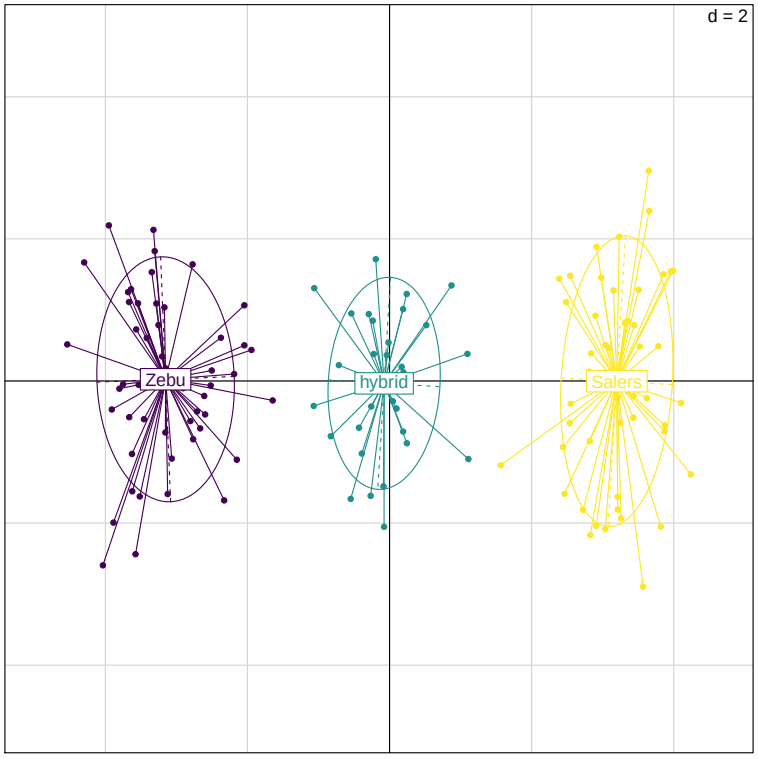
2.2.2 Looking for hybrid using `snapclust`

`snapclust` can also be used in the presence of hybridization. In this case, it will attempt to classify individuals into parental populations (later noted A and B), or various types of hybrids defined by the user. To activate the detection of hybrids, simply specify `hybrids = TRUE`, which will by default look for F1, i.e. 50% of genes coming from each parental population. Other types of hybrids can be specified through the argument `hybrid.coef`, which is a vector of *parental coefficients*. Each value in this vector defines a given type of hybrid, by the proportion of its gene pool contributed by the first parental population. Complementary coefficients will be added automatically for the second parental population. For instance, `hybrid.coef = c(0.25, 0.5)` will be converted into `hybrid.coef = c(0.75, 0.5, 0.25)`, respectively corresponding to backcross with A, F1, and backcross with B.

2.2.3 Looking for first-generation hybrids (F1)

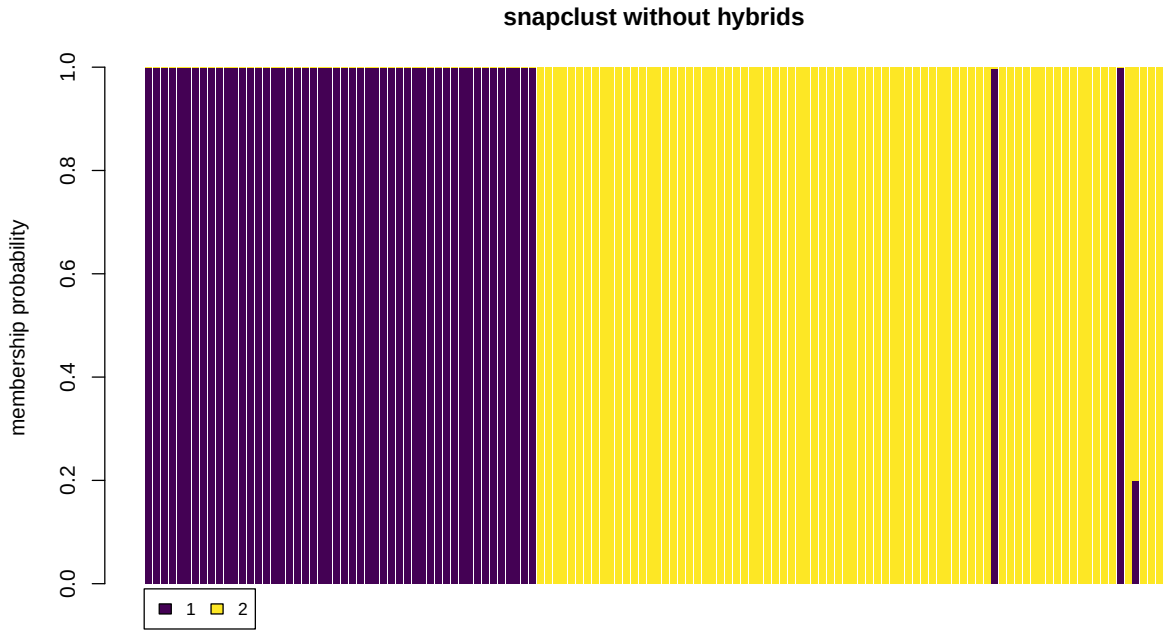
We first use a Principal Component Analysis (PCA) to visualise the simulated hybrid data:

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



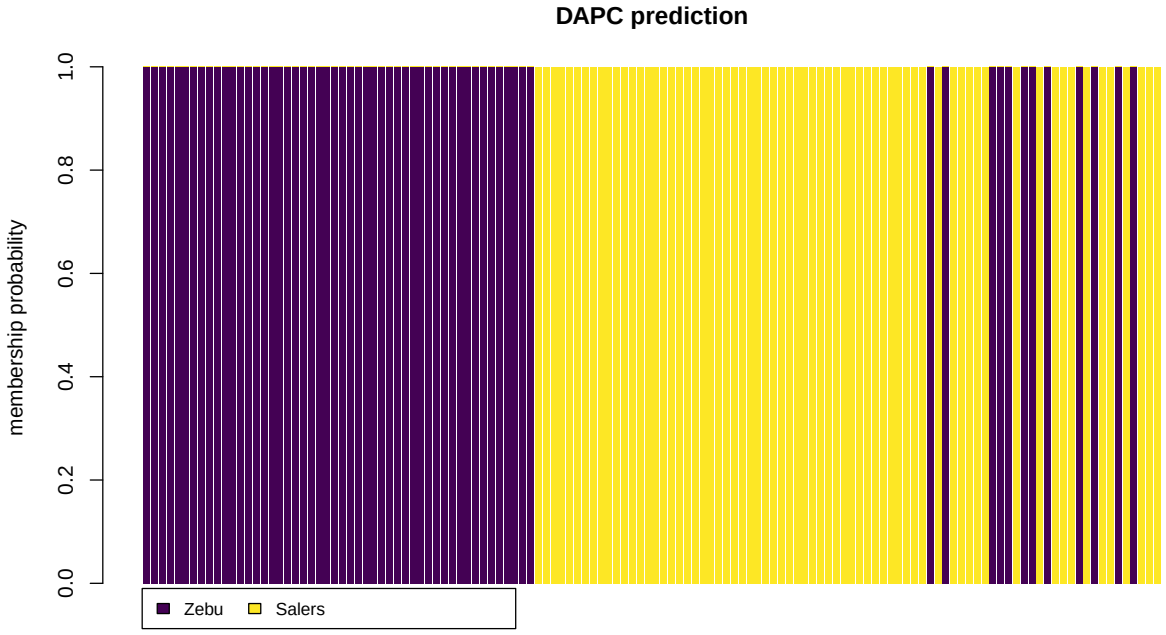
The method applied without hybrid detection correctly identifies the parental populations, but fails to classify hybrids:

```
res.no.hyb <- snapclust(x, k = 2, hybrids = FALSE)
compoplot(res.no.hyb, n.col = 2, col.pal = hybridpal(),
          main = "snapclust without hybrids")
```



One may expect that hybrids would appear as individuals with group membership probabilities around 50% for each parental population. Interestingly, this is almost never the case, irrespective of the type of clustering method. The detail of likelihood contribution (results not shown) suggest that individuals slightly closer to either parental population are simply much more likely to belong to this population. The same can be observed in the Discriminant Analysis of Principal Components (DAPC), which does not use likelihood but a geometric criteria. This can be illustrated by analysing the parental population (first 100 individuals) and predicting the group membership of the hybrids as supplementary individuals:

```
x.dapc <- dapc(x[1:100], n.pca=80, n.da=2)
x.pred <- predict(x.dapc)$posterior
hyb.pred <- predict(x.dapc, newdata = x[-(1:100)])$posterior
compoplot(rbind(x.pred, hyb.pred), col.pal = virid,
          main = "DAPC prediction")
```



The problem of the above is that hybrids are seen as effectively outliers from the parental populations, so that their classification is inherently very unstable. To be formally identified, hybrids need to be modelled as a separate population, whose allele frequency distribution is somewhere between the parental populations. This is exactly the approach taken by `snapclust`:

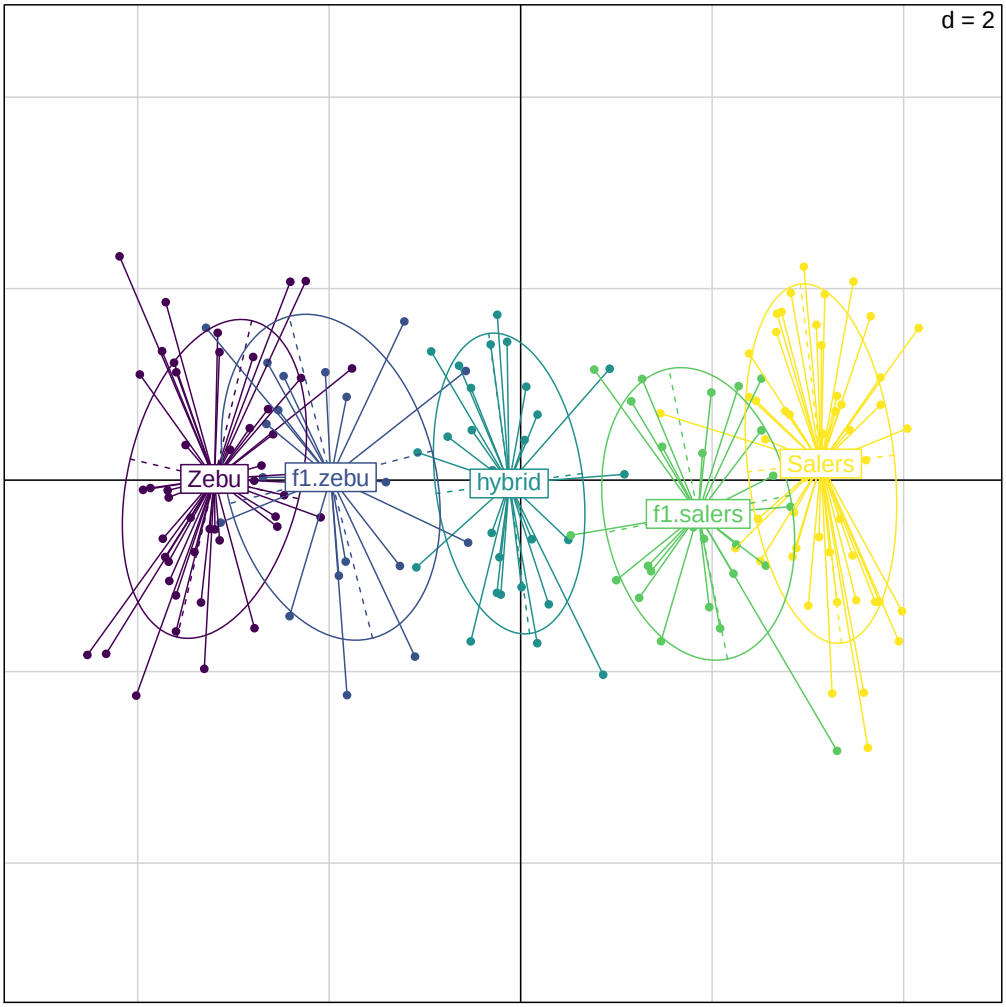
```
res.hyb <- snapclust(x, k = 2, hybrids = TRUE)
compoplot(res.hyb, n.col = 2, col.pal = hybridpal())
```

As we can see, the hybrids are all very well identified.

2.2.4 Looking for F1 and back-crosses

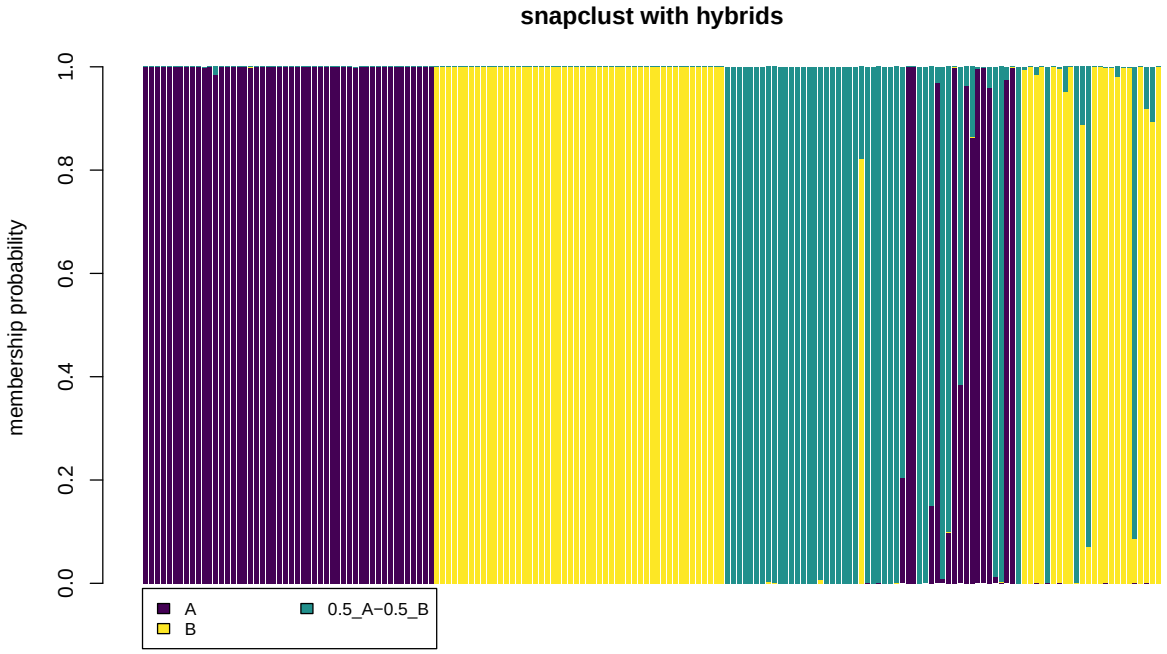
We repeat similar analyses on the dataset `y`, which includes all the individuals from `x` with additional backcrosses. First, a PCA shows the basic structure of the data:

```
y.pca <- dudi.pca(tab(y, NA.method = "mean"), scannf = FALSE, scale = FALSE)
s.class(y.pca$li, pop(y), col = virid(5)[c(1,5,3,2,4)])
```



Looking for F1 hybrids only in `y` shows the expected misclassifications of backcrosses into either hybrids or the closest parental population:

```
res2.hyb <- snapclust(y, k = 2, hybrids = TRUE)
compoplot(res2.hyb, n.col = 2, col.pal = hybridpal(),
          main = "snapclust with hybrids")
```

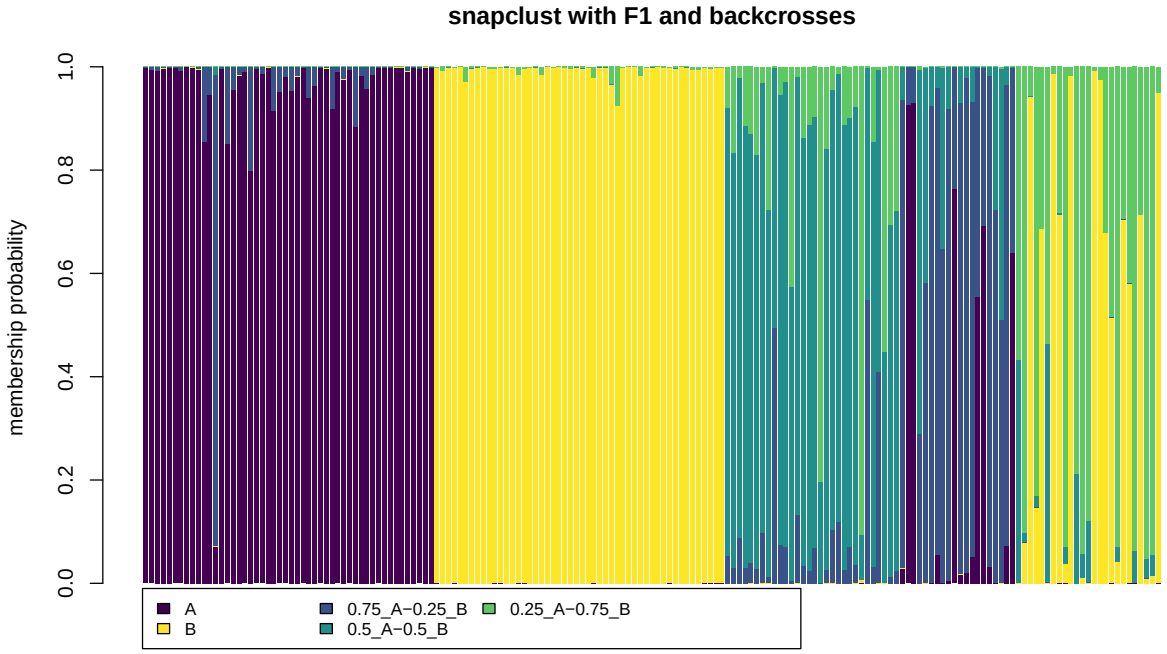


Note that the differentiation between the groups drops very fast, despite a substantial initial F_{st} between the parental populations:

```
hierfstat::pairwise.fst(y)

##             1           2           3           4
## 2 0.11701641
## 3 0.02976662 0.03782376
## 4 0.00889258 0.07194156 0.01456337
## 5 0.06769473 0.01289847 0.01645962 0.04836828
```

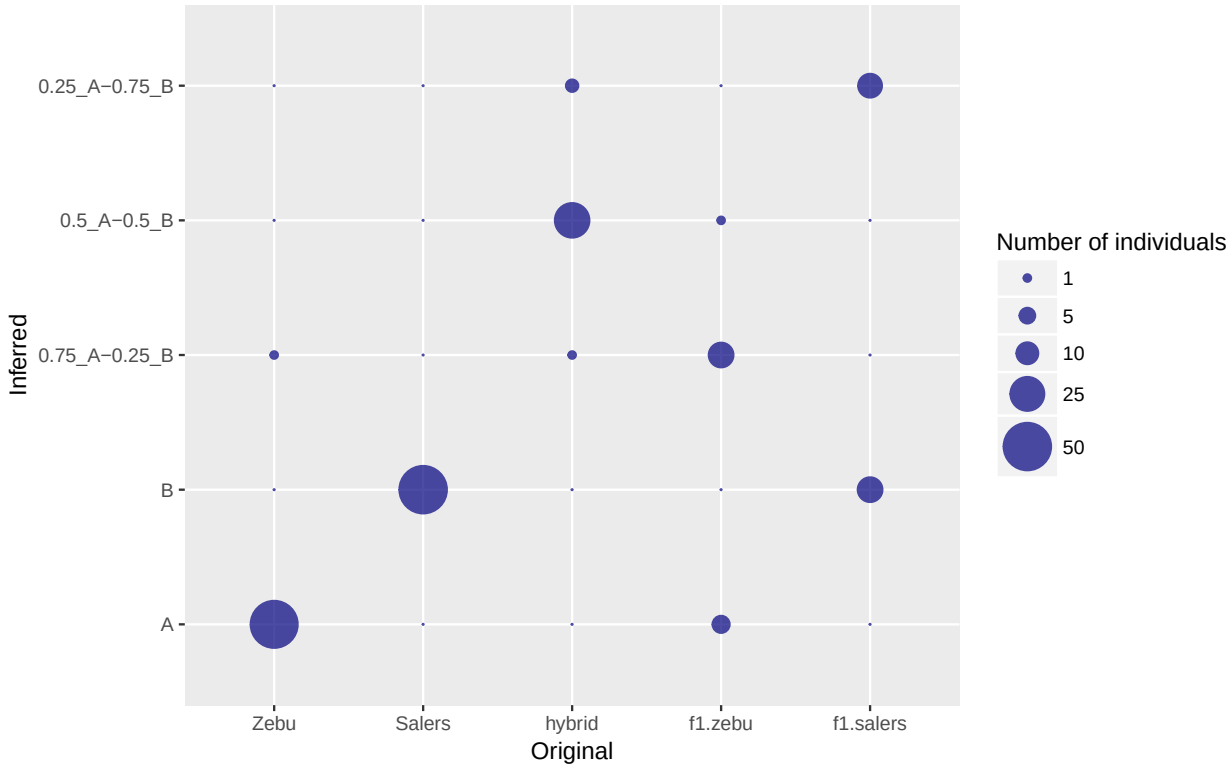
Despite really low differentiation, we still get decent results at identifying backcrosses:

```
res2.back <- snapclust(y, k=2, hybrids = TRUE, hybrid.coef = c(.25, .5))
compoplot(res2.back, col.pal = hybridpal(),
          main = "snapclust with F1 and backcrosses")
```



A number of individuals are mis-classified, but this is merely a reflection of the low differentiation between the backcrosses and the other populations. Let us see the extent of these mis-classifications, using *ggplot2* for an alternative way of displaying this information:

```
library(ggplot2)
tab <- table(pop(y), res2.back$group)
df <- data.frame(tab)
colnames(df) <- c("Original", "Inferred", "Frequency")
p <- ggplot(df, aes(x = Original, y = Inferred)) +
    geom_point(aes(size = Frequency), col="navy", alpha=.7) +
    scale_size_continuous("Number of individuals",
                          breaks = c(1, 5, 10, 25, 50),
                          range = c(0,10))
print(p)
```



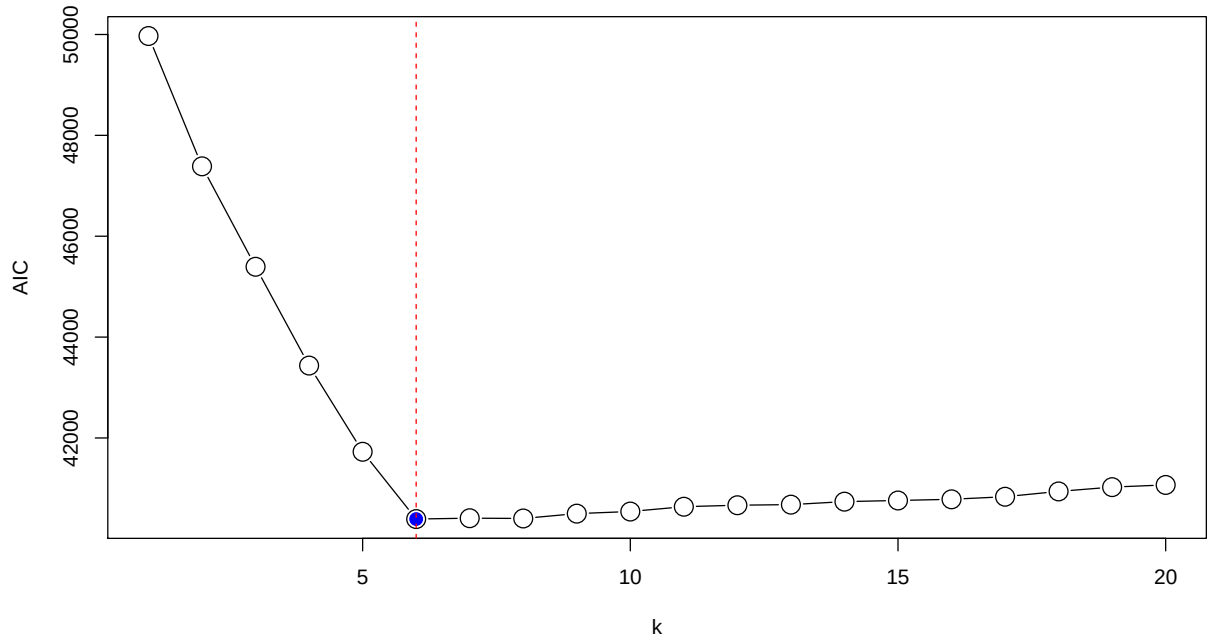

```
tab

##
##               A  B 0.75_A-0.25_B 0.5_A-0.5_B 0.25_A-0.75_B
##   Zebu       49  0             1           0             0
##   Salers      0 50             0           0             0
##   hybrid      0  0             1          26             3
##   f1.zebu     6  0            13           1             0
##   f1.salers   0 13             0           0            12
```

3 Looking for the optimal number of clusters

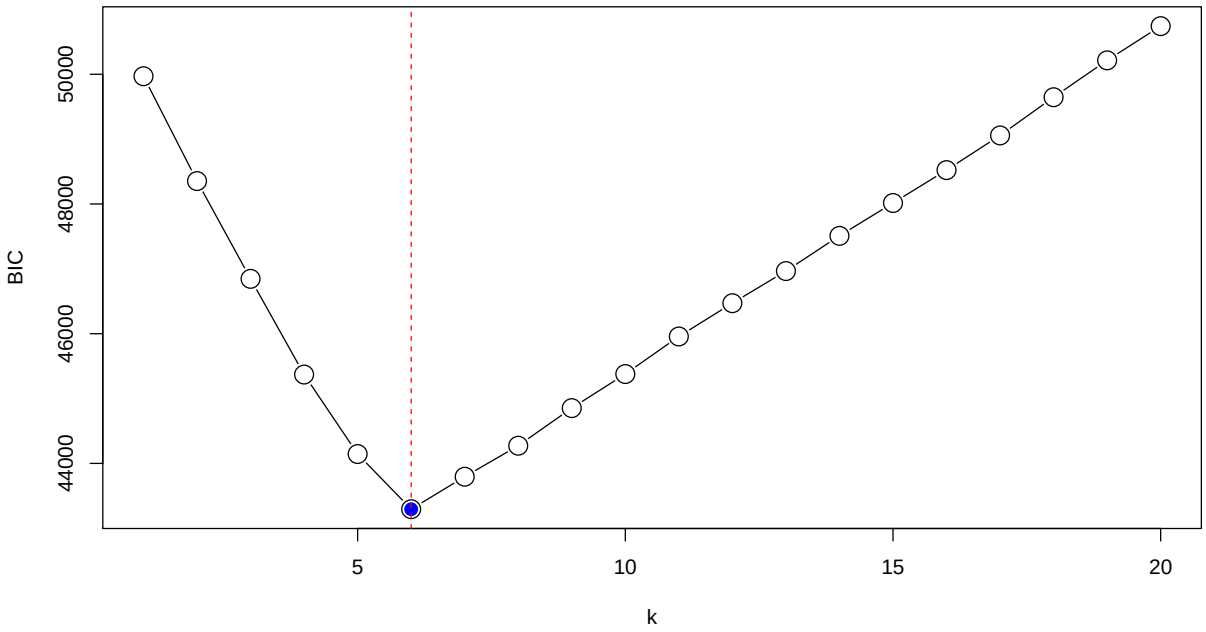
The function `snapclust.choose.k` permits to run `snapclust` for various values of k, and outputs the corresponding AIC or BIC values. Its use is similar to that of `find.clusters`: ideally, the lowest AIC/BIC corresponds to the best model, but in practice sharp decreases in AIC/BIC will reflect substantial improvement of fit. We illustrate it using the `dapcIllus$a` and `dapcIllus$c` datasets, which have 6 and 12 populations, respectively. We first compare the AIC and BIC on `dapcIllus$a`. We indicate the true 'k' by a dashed line, and represent in blue the 'optimal k':

```
data(dapcIllus)
table(pop(dapcIllus$a))
```

```
##
##  P1  P2  P3  P4  P5  P6
## 100 100 100 100 100 100

## look for k = 1:20
a.aic <- snapclust.choose.k(20, dapcIllus$a)
plot(a.aic, type = "b", cex = 2, xlab = "k", ylab = "AIC")
points(which.min(a.aic), min(a.aic), col = "blue", pch = 20, cex = 2)
abline(v = 6, lty = 2, col = "red")
```



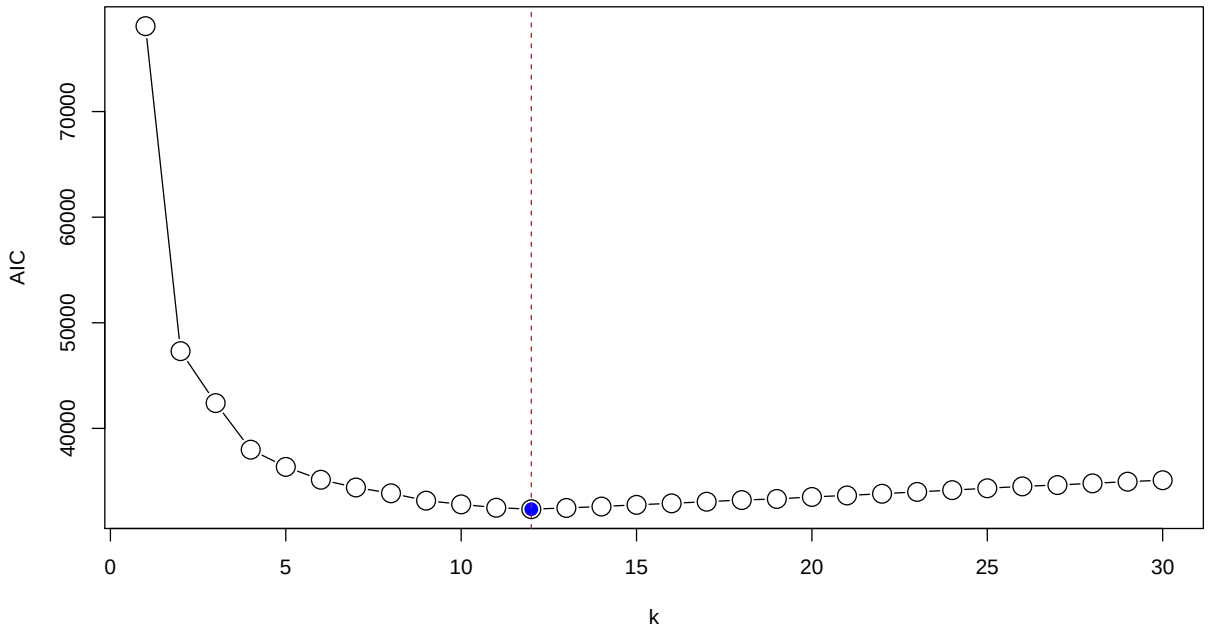

```
## same data, using BIC
a.bic <- snapclust.choose.k(20, dapcIllus$a, IC = BIC)
plot(a.bic, type = "b", cex = 2, xlab = "k", ylab = "BIC")
points(which.min(a.bic), min(a.bic), col = "blue", pch = 20, cex = 2)
abline(v = 6, lty = 2, col = "red")
```



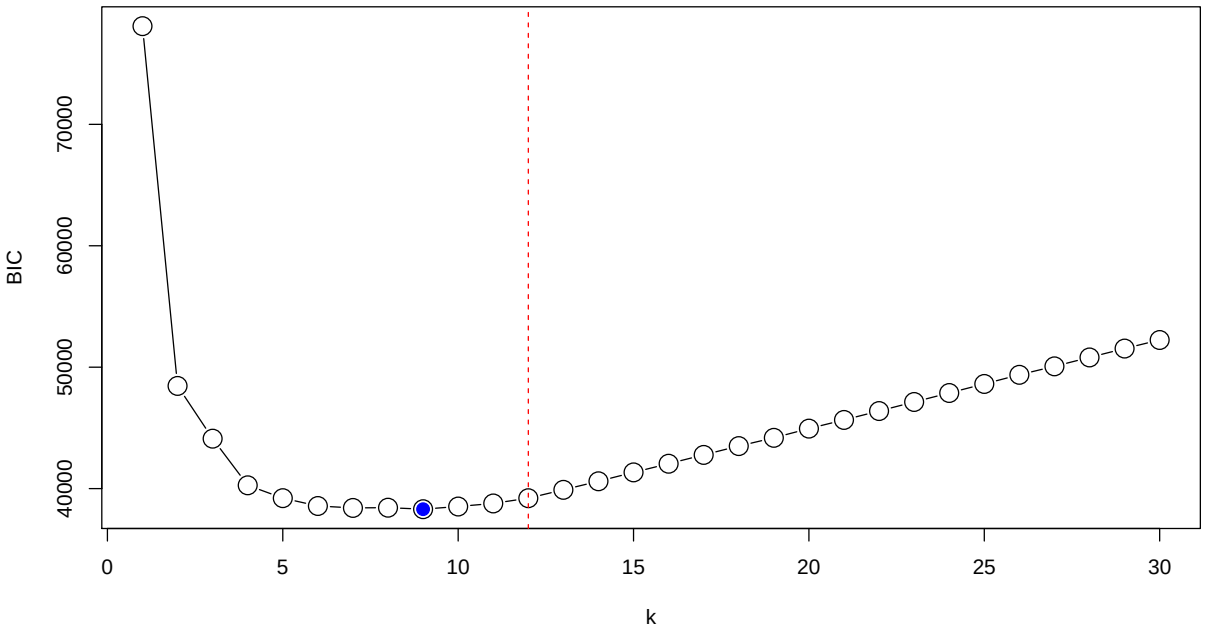
In both cases, the true number of groups (6) is well identified, albeit more clearly by the BIC. Note that this is not systematically the case, and AIC may work better on other data. In practice, it is probably useful to compare both approaches.

We repeat these analyses on the second dataset, which was simulated as a 1-dimensional stepping stone with 12 demes:

```
data(dapcIllus)
table(pop(dapcIllus$c))

##
## P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12
##  50  50  50  50  50  50  50  50  50  50  50  50

## look for k = 1:30
c.aic <- snapclust.choose.k(30, dapcIllus$c)
plot(c.aic, type = "b", cex = 2, xlab = "k", ylab = "AIC")
points(which.min(c.aic), min(c.aic), col = "blue", pch = 20, cex = 2)
abline(v = 12, lty = 2, col = "red")
```




```
## same data, using BIC
c.bic <- snapclust.choose.k(30, dapcIllus$c, IC = BIC)
plot(c.bic, type = "b", cex = 2, xlab = "k", ylab = "BIC")
points(which.min(c.bic), min(c.bic), col = "blue", pch = 20, cex = 2)
abline(v = 12, lty = 2, col = "red")
```


BIC
40000
50000
60000
70000
0
5
10
15
20
25
30
k