Assigning alleles to isoloci in POLYSAT

Lindsay V. Clark
University of Illinois, Urbana-Champaign

June 18, 2016

1 Introduction

This tutorial accompanies the R package POLYSAT versions 1.5 and later, and demonstrates how to use functions in POLYSAT for recoding data from allopolyploid or diploidized autopolyploid organisms such that each microsatellite marker is split into multiple isoloci. The data can then be analyzed under the model of random, Mendelian segregation; for example an allotetraploid organism can be treated as diploid, giving the user access to a much greater number of analyses both with POLYSAT and with other software.

If you are not sure whether your organism is fully polysomic or whether it has two or more subgenomes that segregate independently from each other, keep reading. In the "Quality of clustering" section I illustrate what the results look like if a locus (Loc7 in the example dataset) segratates in a polysomic (autopolyploid) manner.

The methods described in this tutorial are designed for natural populations of one hundred or more individuals. They will also work for certain types of mapping populations, such as an F2 population derived from two inbred grandparents. They will be much less effective for small sample sizes (< 50 for tetraploids, < 200 for higher ploidies), and for datasets with multiple species or highly-diverged ecotypes. The methods in this tutorial are also inappropriate for species that reproduce asexually. If you have duplicate genotypes in your dataset (either as a result of asexual reproduction, vegatative spread, or sampling the same individual multiple times) you should remove them before starting.

It is assumed that the reader is already familiar with R and with POLYSAT. If you aren't, please spend some time with "An Introduction to R" and with the "polysat version 1.5 Tutorial Manual". Additionally, there is a manuscript on BioRxiv at `http://dx.doi.org/10.1101/020610` that describes in detail the rationale and limitations of the allele assignment tools described in this manual. You may also find the documentation for individual functions to be helpful (*e.g.* see `?alleleCorrelations`, `?processDatasetAllo`, and `?recodeAllopoly`). I'm always happy to answer questions over email, but I greatly appreciate it if you have taken the time first to think carefully about your study system

and dataset, understand the methodology and assumptions of any software that you want to use, check your work thoroughly, and in the case of problems with R, consult with someone at your own institution who is experienced with R.

2 Data hygiene

Below I have some recommendations for generating and cleaning up datasets so that they will have the greatest success with the allele assignment algorithm.

2.1 Before you begin genotyping

Choose your markers well. If your research group has previously run microsatellite markers on your species of interest, choose markers that have given clear, consistent, and easy-to-interpret patterns of amplification in the past. If a linkage map has been published for your species, use markers that are on the map; low quality markers would have given segregation distortion and would not have been mappable. Lastly, dinucleotide repeat markers should be used with caution, as their high mutation rate increases the chance of homoplasy occuring, and their high degree of stutter can make amplification patterns difficult to interpret in polyploids. Markers with trinucleotide or larger repeats will give cleaner results, despite not having as many alleles.

Avoid multiplexing several markers in one PCR reaction. Multiplexing increases the probability of scoring error due to allelic dropout. Run each marker in a separate reaction, then, if desired for your electrophoresis method, pool the reactions post-PCR.

Use the highest resolution electrophoresis method available to you. Agarose gels may lack the resolution to distinguish all alleles from each other. If using acrylamide gels, make an allelic ladder by pooling PCR products from several diverse individuals, then run the same ladder on each gel to ensure consistency of scoring. If using a capillary sequencer, always use the same size standard, and be consistent in terms of which fluorescent dye goes with which marker.

"Oh no! I've already run all of my microsatellite markers, and I'm on the last semester of my research assistantship and I need to graduate, so I don't have time to re-do them!" It's okay. Everything above was just a suggestion to improve the probability that the method described in this vignette will work on your dataset. The method may still work, and if it doesn't, you can consider whether failing to meet one of the above suggestions could have caused the problem.

2.2 Scoring your microsatellite alleles

If you know someone in your lab or at your institution who has a lot of experience scoring microsatellites (and if you are not very experienced), get them to sit down with you for a couple hours and demonstrate how they would score

the markers in your dataset. Understanding the additivity of overlapping stutter and allele peaks is important, as is knowing how to distiguish true alleles from PCR artifacts and dye blobs, and knowing how to check that the software interpreted the size standard correctly.

Don't trust any piece of software to score your markers. Most of them were optimized for diploid species, and even then they have a lot of problems. After the software (e.g. GeneMapper or STRand) has called the alleles, you need to manually inspect every genotype, and you will probably correct a lot of them.

Consistency is crucial. One allele may give a pattern of multiple peaks, so you need to decide which peak to score, and whether to round up or down to get the size in nucleotides. If using software that performs "binning", go through and correct all of the allele calls before using them to make bins. Using STRand, I like to take screenshots to indicate how I score each marker, then I can easily look at them months later when I genotype additional individuals.

2.3 Preliminary analysis of the data

In this section I'll use a simulated dataset to demonstrate how to clean up your data and split it into subpopulations if necessary. The same simulated dataset will be used throughout the rest of this manual.

```
> library(polysat)
> data(AllopolyTutorialData)
> summary(AllopolyTutorialData)

Dataset with allele copy number ambiguity.
Simulated allotetraploid dataset.
Number of missing genotypes: 5
303 samples, 7 loci.
1 populations.
Ploidies: NA
Length(s) of microsatellite repeats: 3 4 5

> # make a copy of the dataset to modify
> mydata <- AllopolyTutorialData
```

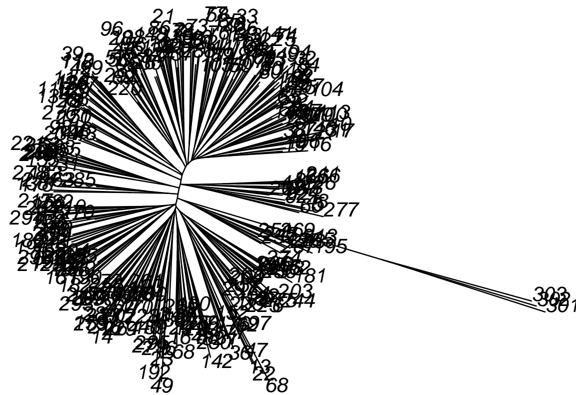
Note that datasets can be imported using any of the normal import functions for POLYSAT (like `read.GeneMapper`). The `data` function here is used only because this is an example dataset installed with the package.

First, any questionable genotypes should be removed. If an electropherogram or banding pattern was unclear, it is best to replace that genotype with missing data. Any duplicate or highly similar genotypes that likely represent the same individual (or a group of asexually derived individuals) should be removed, such that each genotype is only represented once in the dataset. (The `assignClones` function might be useful for identifying duplicates if you haven't already done so.) Since this is an allotetraploid, let's also make sure that no genotypes have more than four alleles, and eliminate any that do.

```
> # Calculate the length of each genotype vector (= the number of alleles) and
> # construct a TRUE/FALSE matrix of whether that number is greater than four.
> tooManyAlleles <- apply(Genotypes(mydata), c(1,2),
+                         function(x) length(x[[1]])) > 4
> # Find position(s) in the matrix that are TRUE.
> which(tooManyAlleles, arr.ind=TRUE) # 43rd sample, second locus

   row col
43  43   2

> # Look at the identified genotype, then replace it with missing data.
> Genotype(mydata, 43, 2)

[1] 143 146 149 152 161

> Genotype(mydata, 43, 2) <- Missing(mydata)
> Genotype(mydata, 43, 2)

[1] -9
```

Next, we'll want to look at population structure in the dataset. We'll make a square matrix of genotype dissimilarities using a simple band-sharing metric, then make a neighbor-joining tree.

```
> mydist <- meandistance.matrix(mydata, distmetric=Lynch.distance,
+                               progress=FALSE)

> require(ape)
> mynj <- nj(mydist)
> plot(mynj, type="unrooted")
```

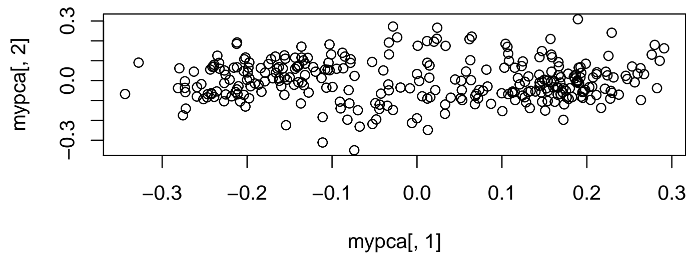


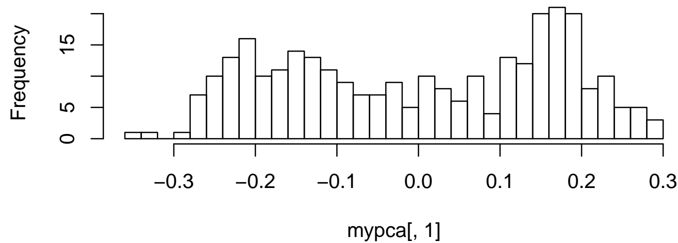
You can see that individuals 301, 302, and 303 are highly dissimilar from the rest. This is what it looks like when some individuals are a different species. Because of the fast rate at which microsatellites mutate, allele assignments that we make in one species are very unlikely to apply to another species. We will remove these three individuals from the dataset.

```
> mydata <- deleteSamples(mydata, c("301","302","303"))
```

Now let's examine the rest of the dataset for population structure using principal coordinates analysis.

```
> par(mfrow=c(2,1))
> mypca <- cmdscale(mydist[Samples(mydata), Samples(mydata)])
> plot(mypca[,1], mypca[,2])
> hist(mypca[,1], breaks=30)
```





We can see a slightly bimodal distribution of individuals, indicating moderate population structure. Since population structure can interfere with allele assignment, we will assign individuals to two populations that we can analyze separately.

```
> pop1ind <- Samples(mydata)[mypca[,1] <= 0]
> pop2ind <- Samples(mydata)[mypca[,1] > 0]
> PopInfo(mydata)[pop1ind] <- 1
> PopInfo(mydata)[pop2ind] <- 2
```

Of course, principal coordinates analysis is not the only method of assigning individuals to populations. If you have already analyzed your data with other software (for example, Structure) before importing it into POLYSAT, you could instead use population assignments from that software, and construct the `PopInfo` vector accordingly to indicate population assignments.

3 The polysat algorithm for allele assignment

3.1 General considerations and parameters

To assign alleles to isoloci, you will primarily be using the function `processDatasetAllo`. This function internally calls two other functions from POLYSAT: `alleleCorrelations` looks for negative correlations between alleles and uses those correlations to

make preliminary assignments, then `testAlGroups` adjusts those assignments if necessary after checking them against individual genotypes. `testAlGroups` has some parameters that affect its accuracy depending on the ploidy of the organism, the size of the population, the rate of meiotic error (pairing between homeologs or paralogs during meiosis), the presence of null alleles, and homoplasy (different alleles with identical amplicon size) between isoloci. Below are some recommendations for adjusting arguments from the defaults depending on particular issues in the dataset.

Ploidy greater than tetraploid	Increase `rare.al.check`
Small sample size	Increase `rare.al.check`
Meiotic error	Increase `tolerance`
Null alleles	`null.weight = 0`
Null alleles at high frequency	`rare.al.check = 0`
Homoplasy	`rare.al.check = 0`

Because you may not know whether the last four issues are present in your dataset, I recomment trying several parameter sets. The function `processDatasetAllo` tests several parameter combinations across all loci in the dataset.

3.2 Running the algorithm

Now we will use `processDatasetAllo` to make allele assignments. Because this is an allotetraploid we will use `n.subgen = 2` (two subgenomes) and `SGploidy = 2` (each subgenome is diploid). Because we divided our dataset into two subpopulations that appear to have different allele frequencies, we will set `usePops = TRUE` to analyze the two populations separately according to `PopInfo(mydata)`. We will leave the `parameters` argument at the default, which includes one parameter set optimized for no alleles and no homoplasy, one optimized for homoplasy, and two optimized for null alleles.

```
> myassign <- processDatasetAllo(mydata, n.subgen = 2, SGploidy = 2,
+                                usePops = TRUE)
```

```
Warning: Significant positive correlations between alleles at locus Loc6 ; population struct
Warning: Significant positive correlations between alleles at locus Loc6 ; population struct
```

3.3 Inspecting the results

3.3.1 Warnings about positive correlations

We got a warning about Loc6 for both the populations. If population structure were a serious problem, we would have gotten warnings about most or all loci. Let's take a look at which alleles had positive correlations for Loc6, to see if there may have been a scoring problem.

The `myassign` object is a list, and it has an element called `AlCorrArray` that contains the results of `alleleCorrelations` for each locus and population.

Each set of results is also a list. The list element called `significant.pos` indicates any significantly positive associations between alleles. We will look at the Loc6 results for both populations.

```
> myassign$AlCorrArray[["Loc6", 1]]$significant.pos

      300   303   306   327   330   336   339   345   348
300    NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
303 FALSE    NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE
306 FALSE FALSE    NA FALSE FALSE FALSE FALSE FALSE FALSE
327 FALSE FALSE FALSE    NA  TRUE FALSE FALSE FALSE FALSE
330 FALSE FALSE FALSE  TRUE    NA FALSE FALSE FALSE FALSE
336 FALSE FALSE FALSE FALSE FALSE    NA  TRUE FALSE FALSE
339 FALSE FALSE FALSE FALSE FALSE  TRUE    NA FALSE FALSE
345 FALSE FALSE FALSE FALSE FALSE FALSE FALSE    NA  TRUE
348 FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE    NA
```

```
> myassign$AlCorrArray[["Loc6", 2]]$significant.pos

      300   303   306   327   330   336   339   345   348
300    NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
303 FALSE    NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE
306 FALSE FALSE    NA FALSE FALSE FALSE FALSE FALSE FALSE
327 FALSE FALSE FALSE    NA  TRUE FALSE FALSE FALSE FALSE
330 FALSE FALSE FALSE  TRUE    NA FALSE FALSE FALSE FALSE
336 FALSE FALSE FALSE FALSE FALSE    NA  TRUE FALSE FALSE
339 FALSE FALSE FALSE FALSE FALSE  TRUE    NA FALSE FALSE
345 FALSE FALSE FALSE FALSE FALSE FALSE FALSE    NA  TRUE
348 FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE    NA
```

We see positive correlations between alleles 330 and 327, 339 and 336, and 348 and 345. Since these are trinucleotide repeats, it looks like some of the larger alleles (which would tend to have more stutter) had stutter peaks miscalled as alleles. If this were a real dataset, I would say to go back to the gels or elecropherograms and call the alleles more carefully. Since this is a simulated dataset, we will simply exclude Loc6 from further analysis.

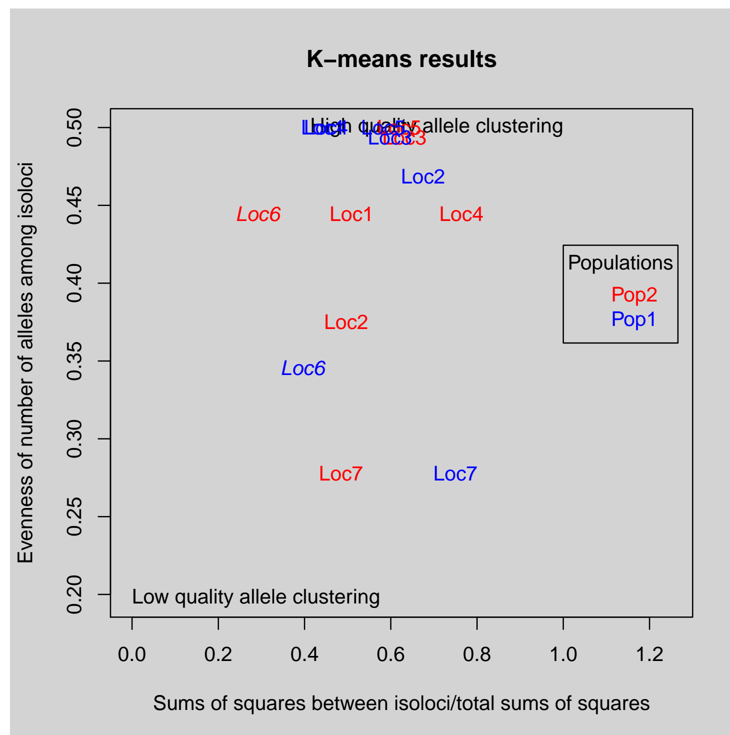
```
> mydata <- deleteLoci(mydata, loci="Loc6")
```

Occasionally, a locus with no population structure or scoring error will produce a warning about positive correlations between alleles simply due to sampling error in the dataset. If you only get the warning for one locus and population in your dataset, and it does not look like there was scoring error (*i.e.* the alleles that are positively correlated are not close in amplicon size), you can safely ignore the warning.

3.3.2 Quality of clustering

In your working directory, you should now find a file called "alleleAssignment-Plots.pdf" containing several plots to help you evaluate clustering quality. We will recreate some of those plots here and discuss their interpretation.

```
> par(mfrow = c(1,1))
> plotSSAllo(myassign$AlCorrArray)
```



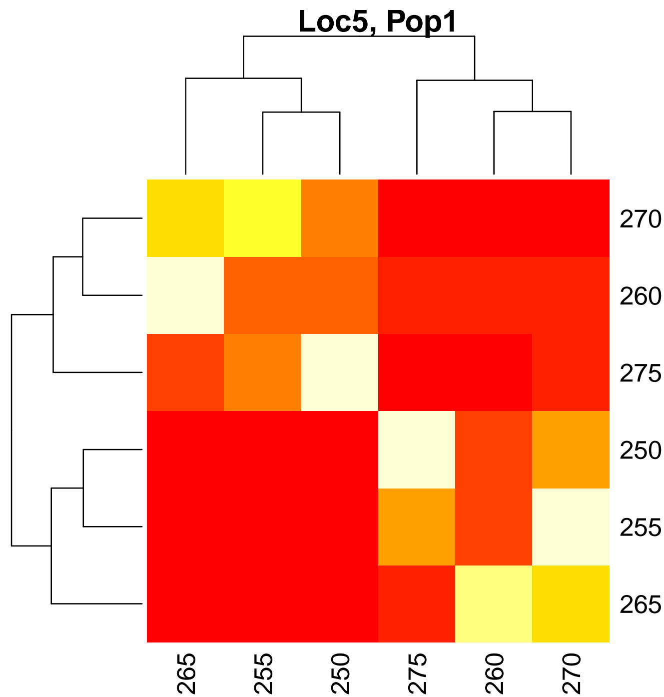
This plot gives an indication of clustering quality using the K-means method (from `alleleCorrelations`) alone, and is intended to help identify problematic loci or populations. The x-axis essentially shows how much of the variation in P-values for correlations between alleles could be explained by clustering alleles into groups. The y-axis indicates how similar in size the groups of alleles are, *e.g* two groups of five alleles each would give a high value, and one group with one allele and one group with nine alleles would give a low value. So, loci and populations in the upper-right quadrant performed well in allele assignment, but others should be subject to scrutiny.

In the last section we identified Loc6 as a locus that we should exclude. In this plot, for both populations, Loc6 has a relatively low value on the x-axis. It is also marked in italics to indicate that there were positive correlations between alleles, in case we hadn't noticed this problem when the algorithm ran.

It looks like Loc7 may also have a problem, given that it got a low value on the y-axis in both populations.

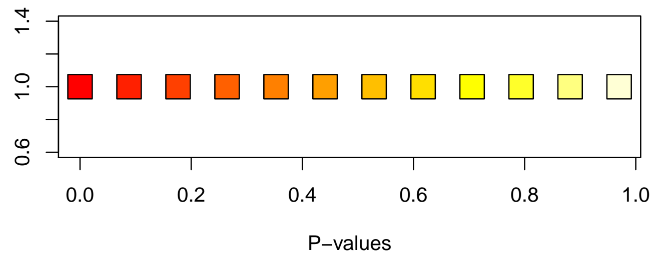
We can also make heatmaps of the P-values for correlations between alleles to get a sense of how well the clustering went. Let's compare a locus that seemed to work very well (Loc5) to one that we expect to cluster poorly (Loc6), then compare them to Loc7.

```
> heatmap(myassign$AlCorrArray[["Loc5", "Pop1"]]$heatmap.dist,
+         main = "Loc5, Pop1")
```

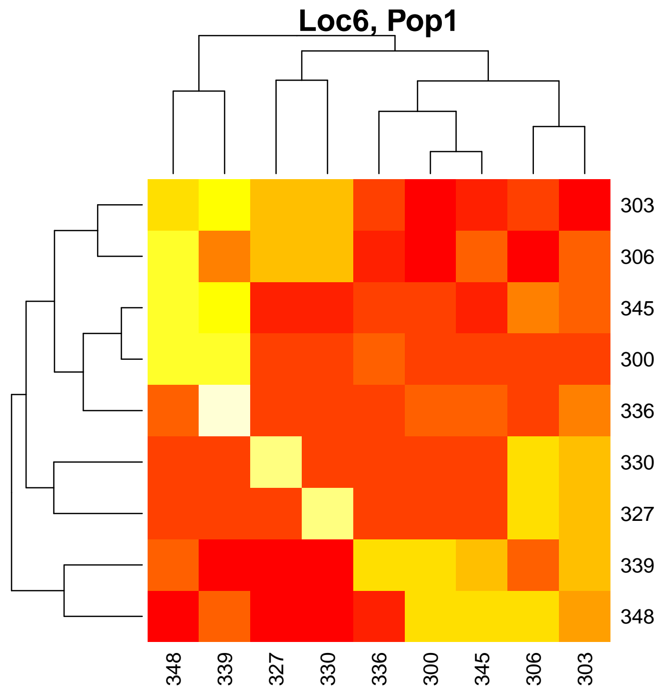


The big red blocks neatly divide the alleles into two groups.

```
> # A plot to show how the colors correspond to p-values in the
> # heat map; you can repeat this for the other heat maps in this
> # tutorial if you wish.
> plot(x=seq(min(myassign$AlCorrArray[["Loc5", "Pop1"]]$heatmap.dist),
+            max(myassign$AlCorrArray[["Loc5", "Pop1"]]$heatmap.dist),
+            length.out=12),
+      y=rep(1,12), xlab="P-values", ylab="", bg=heat.colors(12),
+      pch=22, cex=3)
```

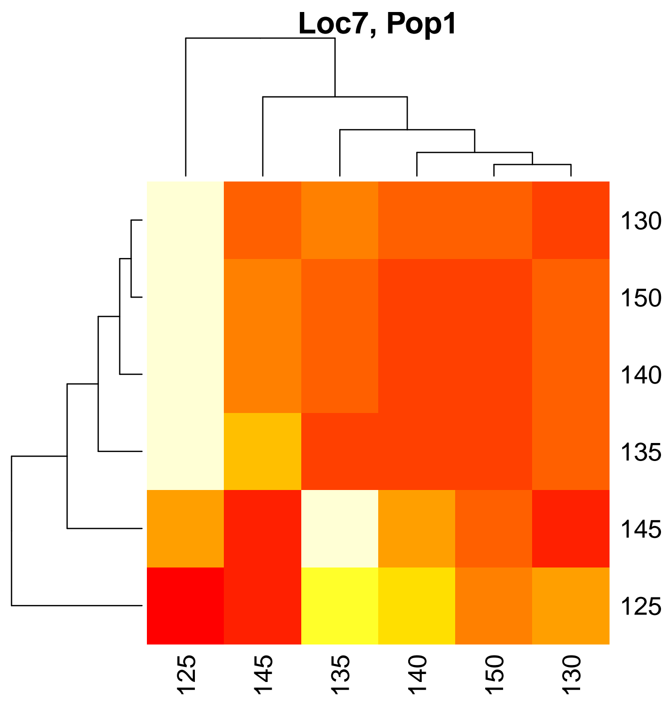



```
> heatmap(myassign$AlCorrArray[["Loc6", "Pop1"]]$heatmap.dist,
+         main = "Loc6, Pop1")
```

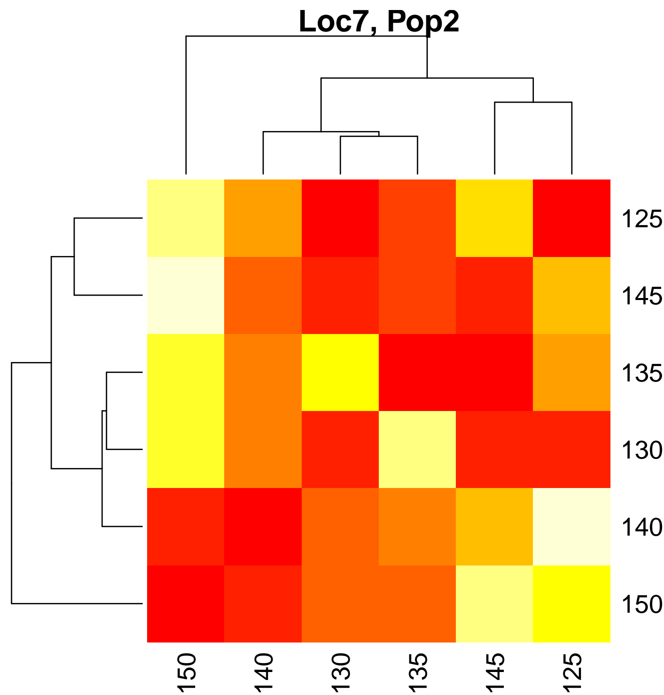


For Loc6, it is not nearly as obvious how to divide the alleles into two groups.

```
> heatmap(myassign$AlCorrArray[["Loc7", "Pop1"]]$heatmap.dist,
+         main = "Loc7, Pop1")
```




```
> heatmap(myassign$AlCorrArray[["Loc7", "Pop2"]]$heatmap.dist,
+         main = "Loc7, Pop2")
```

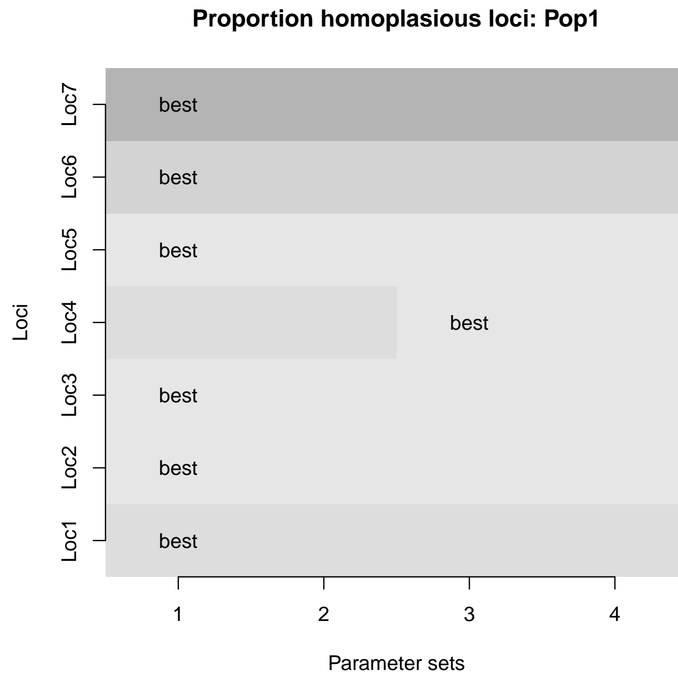


For Loc7, the alleles also don't seem to go into two neat groups. In fact, the dendrograms on both heatmaps first split off a single allele from the rest, but that allele is different for Pop1 (125) versus Pop2 (150). When I simulated Loc7, I actually simulated it as being a single tetrasomic locus instead of a pair of disomic isoloci. If you have a large sample size and no positive correlations between alleles, but the results for all of your loci look like this, you can probably treat the data as being autopolyploid (polysomic). It is also quite possible for an organism to have some disomic loci and some polysomic loci. We will leave Loc7 in the dataset but won't attempt to do allele assignment or genotype recoding with it.

3.4 Evaluating the parameter sets

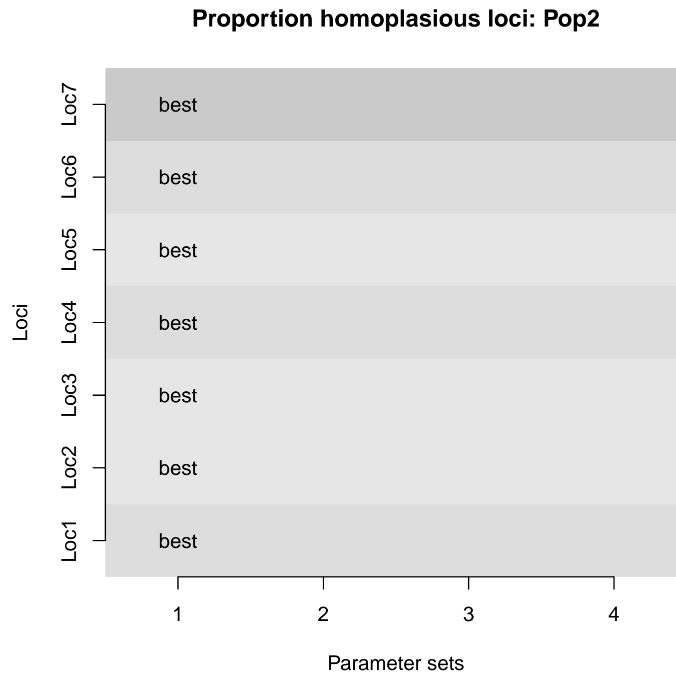
Now we will take a look at the results of `testAlGroups`. As you may recall, there are several parameters that can be adjusted for that function, and we tried out the default four sets of parameters with `processDatasetAllo`. One way to evaluate the quality of the assignments produced by `testAlGroups` is by seeing how many alleles it decided to make homoplasious, *i.e.* how many alleles were assigned to multiple isoloci. Let's plot those proportions separately for the two populations.

```
> plotParamHeatmap(myassign$propHomoplasious, popname = "Pop1",
+                  main = "Proportion homoplasious loci:")
```



Lighter colors indicate fewer homoplasious alleles, which in turn indicates better quality allele assignment. Loc7 had a lot of homoplasious alleles, but that is not a surprise given that it is one tetrasomic locus rather than two disomic loci. In Pop1, Loc4 worked best with the third and fourth parameter sets (allowing null alleles), but the other loci worked equally well regardless of parameter set.

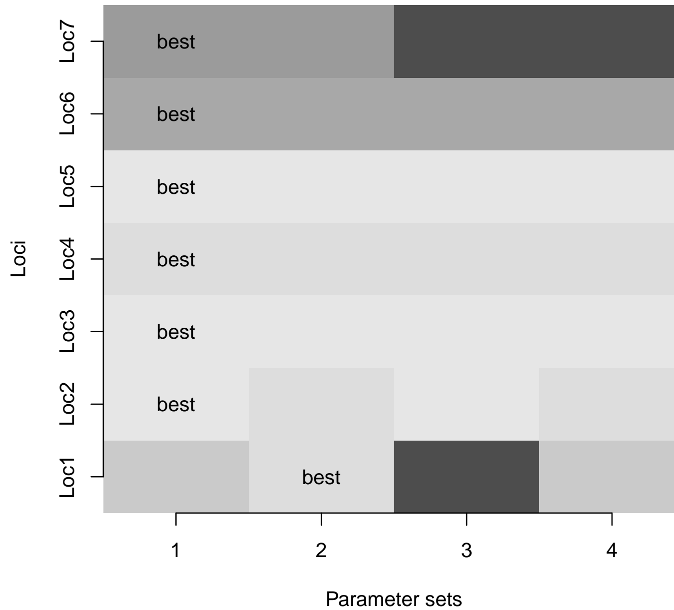
```
> plotParamHeatmap(myassign$propHomoplasious, popname = "Pop2",
+                  main = "Proportion homoplasious loci:")
```



For all loci in Pop2, we see the same proportion of homoplasious loci regardless of parameter set.

`processDatasetAllo` also runs the `mergeAlleleAssignments` function to combine allele assignments across populations, within loci and parameter sets. We take a look at how many more homoplasious alleles we get after merging the assignments (since the same allele may have been assigned to different isoloci in different populations).

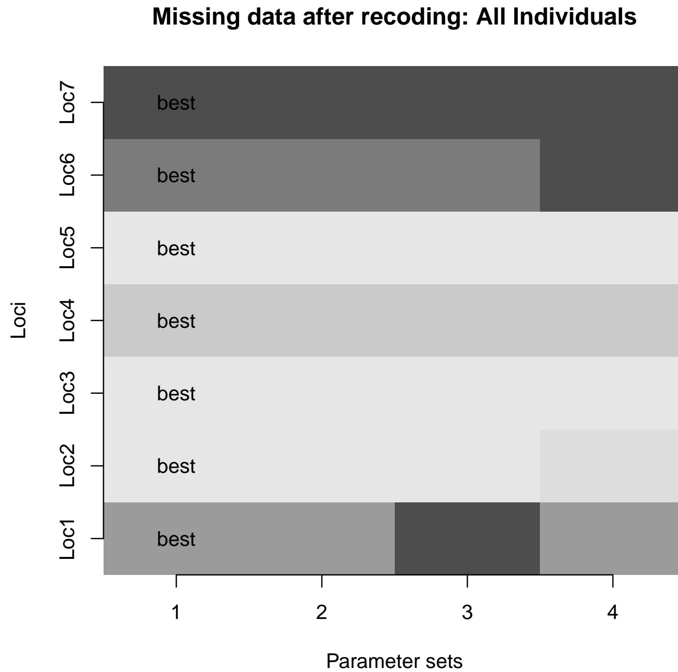
```
> plotParamHeatmap(myassign$propHomoplMerged, popname = "Merged across populations",
+                  main = "Proportion homoplasious loci:")
```



Now we see that Loc6 and Loc7 look a lot worse, since their allele assignments were different between populations. For Loc1, the second parameter set (optimized for homoplasy) was clearly the best in terms of minimizing homoplasious alleles. For Loc2, either the first or third parameter set (the two parameter sets that include the allele swapping algorithm) works best. For the other loci, we have the same number of homoplasious alleles regardless of parameter set.

We can also examine, for each locus and parameter set, what proportion of genotypes would be replaced with missing data if we used that set of allele assignments to recode the data. Genotypes are recoded as missing if homoplasy prevents them from being unambiguously determined. `processDatasetAllo` runs `recodeAllopoly` in order to determine these missing data rates. When it does so, it uses the `allowAneuploidy=FALSE` option so that genotypes that have too many alleles (> `SGploidy`) for an isolocus will also be recoded as missing.

```
> plotParamHeatmap(myassign$missRate, popname = "All Individuals",
+                  main = "Missing data after recoding:")
```



We can see that for Loc3, Loc4, and Loc5, the parameter set does not affect the amount of missing data.

`processDatasetAllo` suggests a best set of allele assignments based on missing data after recoding and the proportion of homoplasious alleles. For each locus, it picks the parameter set that results in the lowest missing data rate, then in the case of a tie the parameter set with the least amount of homoplasy (after merging across populations), then in the case of a tie the lowest numbered parameter set. So in this case, parameter set 2 would be chosen for Loc1, and parameter set 1 for all other loci. (Loc1 had an equal amount of missing data for parameter sets 1 and 2, but less homoplasy for parameter set 2.)

Let's extract that list of optimal assignments from the results. We will also eliminate Loc6 and Loc7 from the list since we discarded Loc6 and don't want to recode Loc7.

```
> myBestAssign <- myassign$bestAssign
> myBestAssign

[[1]]
[[1]]$locus
[1] "Loc1"

[[1]]$SGploidy
[1] 2
```

```
[[1]]$assignments
     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0


[[2]]
[[2]]$locus
[1] "Loc2"

[[2]]$SGploidy
[1] 2

[[2]]$assignments
     140 143 146 149 152 155 158 161
[1,]   1   1   1   1   0   0   1   1
[2,]   0   0   0   0   1   1   0   0


[[3]]
[[3]]$locus
[1] "Loc3"

[[3]]$SGploidy
[1] 2

[[3]]$assignments
     98 102 106 110 114 118 122 126 130
[1,]  0   0   0   0   0   1   1   1   1
[2,]  1   1   1   1   1   0   0   0   0


[[4]]
[[4]]$locus
[1] "Loc4"

[[4]]$SGploidy
[1] 2

[[4]]$assignments
     181 185 189 193 197 201
[1,]   0   0   1   1   1   0
[2,]   1   1   0   1   0   1
```

```
[[5]]
[[5]]$locus
[1] "Loc5"

[[5]]$SGploidy
[1] 2

[[5]]$assignments
     250 255 260 265 270 275
[1,]   0   0   1   0   1   1
[2,]   1   1   0   1   0   0


[[6]]
[[6]]$locus
[1] "Loc6"

[[6]]$SGploidy
[1] 2

[[6]]$assignments
     300 303 306 327 330 336 339 345 348
[1,]   1   1   1   0   1   1   0   0   0
[2,]   1   1   1   1   1   1   1   1   1


[[7]]
[[7]]$locus
[1] "Loc7"

[[7]]$SGploidy
[1] 2

[[7]]$assignments
     125 130 135 140 145 150
[1,]   1   1   1   0   1   1
[2,]   0   1   1   1   1   1
```

```
> myBestAssign <- myBestAssign[1:5]
```

We can see a large amount of homoplasy for Loc6 and Loc7, since the allele assignment algorithm was not appropriate for those two loci. We can also see that Loc1 and Loc4 each have one homoplasious allele. This accounts for the much higher proportion of missing data when recoding Loc1 and Loc4 as opposed to Loc2, Loc3, and Loc5.

It is possible to take a deeper look at the allele assignments that were produced before the best set was chosen. Let's look at Loc1, since it has such a

high missing data rate after recoding.

```
> myassign$AlCorrArray[["Loc1", "Pop1"]]$Kmeans.groups

     224 227 230 233 236 239
[1,]   1   0   1   1   0   0
[2,]   0   1   0   0   1   1

> myassign$AlCorrArray[["Loc1", "Pop2"]]$Kmeans.groups

     224 227 230 233 236 239
[1,]   0   1   0   0   0   1
[2,]   1   0   1   1   1   0

> myassign$AlCorrArray[["Loc1", "Pop1"]]$UPGMA.groups

     224 227 230 233 236 239
[1,]   1   0   1   1   0   0
[2,]   0   1   0   0   1   1

> myassign$AlCorrArray[["Loc1", "Pop2"]]$UPGMA.groups

     224 227 230 233 236 239
[1,]   1   0   1   1   1   0
[2,]   0   1   0   0   0   1
```

Both K-means clustering and UPGMA placed allele 236 in a different isolocus depending on population. This is the same allele that was ultimately assigned to be homoplasious in the "best" assignment set.

`testAlGroups` results for Loc1:

```
> myassign$TAGarray[["Loc1", "Pop1", 1]]$assignments

     224 227 230 233 236 239
[1,]   1   0   1   1   1   0
[2,]   0   1   0   0   1   1

> myassign$TAGarray[["Loc1", "Pop2", 1]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   1   1   1
[2,]   1   0   1   0   1   0

> myassign$mergedAssignments[["Loc1", 1]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   1   1   1
[2,]   1   0   1   1   1   0
```

```
> myassign$TAGarray[["Loc1", "Pop1", 2]]$assignments

     224 227 230 233 236 239
[1,]   1   0   1   1   1   0
[2,]   0   1   0   0   1   1

> myassign$TAGarray[["Loc1", "Pop2", 2]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0

> myassign$mergedAssignments[["Loc1", 2]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0

> myassign$TAGarray[["Loc1", "Pop1", 3]]$assignments

     224 227 230 233 236 239
[1,]   1   1   1   1   0   0
[2,]   0   1   0   0   1   1

> myassign$TAGarray[["Loc1", "Pop2", 3]]$assignments

     224 227 230 233 236 239
[1,]   1   1   0   1   0   1
[2,]   1   0   1   0   1   0

> myassign$mergedAssignments[["Loc1", 3]]$assignments

[1] "No assignment"

> myassign$TAGarray[["Loc1", "Pop1", 4]]$assignments

     224 227 230 233 236 239
[1,]   1   1   1   1   0   0
[2,]   0   1   0   0   1   1

> myassign$TAGarray[["Loc1", "Pop2", 4]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0

> myassign$mergedAssignments[["Loc1", 4]]$assignments

     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   1   1   1   1   0
```

We can see that with parameter set 1, the swapping of allele 233 to a different isolocus in Pop2 caused the increased homoplasy when assignments were merged across populations. With parameter set 2, however, assignments were consistent between the two populations. Parameter set 3 produced allele assignments that were very different from those produced by parameter sets 1 and 2. Moreover, the assignments produced by parameter set 3 were so different between the two populations that it was not possible to merge them. The parameter set 4 assignments matched the "best" assignment set for Pop2, but not Pop1.

3.5 Re-analyzing individual loci

For problematic loci, you may wish to try different population splits or parameter sets. It is not necessary to re-run `processDatasetAllo` on the entire dataset in order to accomplish this; we can run `alleleCorrelations` and `testAlGroups` on individual loci instead.

Continuing to examine Loc1, what assignments do we get from K-means clustering and UPGMA if we don't split the datset into two populations?

```
> corrLoc1AllInd <- alleleCorrelations(mydata, locus = "Loc1", n.subgen = 2)
> corrLoc1AllInd$Kmeans.groups
```

```
     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   0   0
```

```
> corrLoc1AllInd$UPGMA.groups
```

```
     224 227 230 233 236 239
[1,]   1   0   1   1   0   0
[2,]   0   1   0   0   1   1
```

Analyzing both populations together gives us clustering identical to the previous clustering from Pop1.

We'll try `testAlGroups` with a parameter set identical to parameter set 2 from our previous analysis. We'll also try increasing `tolerance` a little bit to see if it eliminates homoplasy. Increasing `tolerance` could be useful if Loc1 does not in fact segregate in a completely disomic manner.

```
> TaLoc1.param2 <- testAlGroups(mydata, corrLoc1AllInd, SGploidy = 2,
+                               null.weight = 0.5, tolerance = 0.05,
+                               rare.al.check = 0)
> TaLoc1.param5 <- testAlGroups(mydata, corrLoc1AllInd, SGploidy = 2,
+                               null.weight = 0.5, tolerance = 0.1,
+                               rare.al.check = 0)
> TaLoc1.param2$assignments
```

```
     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0
```

```
> TaLoc1.param5$assignments
```

```
     224 227 230 233 236 239
[1,]   0   1   0   0   1   1
[2,]   1   0   1   1   1   0
```

In both cases we get the same allele assignments as before, so we will leave it as is.

Hypothetically though, if we liked the new assignments better and wanted to use them to replace the old assignments, here is how it would be done:

```
> myBestAssign[[1]] <- TaLoc1.param5
```

3.6 Recoding the data

Now that we've chosen sets of allele assignments to use and thrown away loci that had problems, we can recode the dataset.

```
> recodedData <- recodeAllopoly(mydata, myBestAssign)
> summary(recodedData)
```

```
Dataset with allele copy number ambiguity.
Simulated allotetraploid dataset.
Number of missing genotypes: 520
300 samples, 11 loci.
2 populations.
Ploidies: 2 3 1 NA
Length(s) of microsatellite repeats: 3 4 5
```

You'll notice that we have more loci now that each marker (except Loc7) has been split into two isoloci. We also have a lot of missing data, since homoplasy can lead to uncertainty about what the true genotype is. We had homoplasy for Loc1 and Loc 4.

```
> for(L in Loci(recodedData)){
+   proportionmissing <- mean(isMissing(recodedData, loci=L))
+   cat(paste(L,":",proportionmissing,"missing"),sep="\n")
+ }
```

```
Loc1_1 : 0.513333333333333 missing
Loc1_2 : 0.52 missing
Loc2_1 : 0.00333333333333333 missing
Loc2_2 : 0.00333333333333333 missing
Loc3_1 : 0.00333333333333333 missing
Loc3_2 : 0.00333333333333333 missing
Loc4_1 : 0.323333333333333 missing
Loc4_2 : 0.343333333333333 missing
Loc5_1 : 0.01 missing
Loc5_2 : 0.01 missing
Loc7 : 0 missing
```

You'll also notice that not the entire dataset is diploid. That is because there is some meiotic error in the dataset, and we used `allowAneuoploidy = TRUE` in `recodeAllopoly`. Most genotypes are diploid though.

```
> table(Ploidies(recodedData))

   1    2    3
   3 2994    3
```

We can manually add in the ploidy for Loc7, since it was not recoded.

```
> Ploidies(recodedData)[,"Loc7"] <- 4
```

The recoded data may now be analyzed with any POLYSAT function, or exported to other software using any of the various `write` functions in polysat. For example, it is now appropriate to estimate allele frequencies from the data and use those to estimate G_{ST}.

```
> myfreq <- simpleFreq(recodedData)
> myGst <- calcPopDiff(myfreq, metric = "Gst")
> myGst

             Pop2         Pop1
Pop2 -0.002039396  0.039108231
Pop1  0.039108231 -0.002151562
```

We can also export the data if we want to do analysis using other software.

```
> write.GeneMapper(recodedData, file = "tutorialRecodedData.txt")
```

4 The Catalán method of allele assignment

An alternative method of allele assignment available in POLYSAT is that by Catalán *et al.* (2006; `http://dx.doi.org/10.1534/genetics.105.042788`). The Catalán method does not allow for homoplasy, null alleles, or meiotic error, but may perform better than the POLYSAT method in cases of strong population structure. Let's try it on our example dataset.

```
> catResults <- list()
> length(catResults) <- length(Loci(mydata))
> names(catResults) <- Loci(mydata)
> for(L in Loci(mydata)){
+   cat(L, sep="\n")
+   catResults[[L]] <- catalanAlleles(mydata, locus=L, verbose=TRUE)
+ }
```

```
Loc1
$locus
[1] "Loc1"

$SGploidy
[1] 2

$assignments
[1] "Homoplasy or null alleles: some genotypes have too few alleles"

Loc2
Allele assignments:
     140 143 146 149 152 155 158 161
[1,]   0   0   0   0   1   1   0   0
[2,]   1   1   1   1   0   0   1   1
Inconsistent genotypes:
[[1]]
[1] 143 155 158 161

[[2]]
[1] 140 143 149 155

$locus
[1] "Loc2"

$SGploidy
[1] 2

$assignments
[1] "Homoplasy or null alleles"

Loc3
Allele assignments:
     98 102 106 110 114 118 122 126 130
[1,]  1   1   1   1   1   0   0   0   0
[2,]  0   0   0   0   0   1   1   1   1
Inconsistent genotypes:
[[1]]
[1] 102 106 110 118

$locus
[1] "Loc3"

$SGploidy
[1] 2
```

```
$assignments
[1] "Homoplasy or null alleles"

Loc4
$locus
[1] "Loc4"

$SGploidy
[1] 2

$assignments
[1] "Homoplasy or null alleles: some genotypes have too few alleles"

Loc5
$locus
[1] "Loc5"

$SGploidy
[1] 2

$assignments
     250 255 260 265 270 275
[1,]   1   1   0   1   0   0
[2,]   0   0   1   0   1   1

Loc7
$locus
[1] "Loc7"

$SGploidy
[1] 2

$assignments
[1] "Homoplasy or null alleles: some genotypes have too few alleles"
```

Assignments are returned for Loc5 only. For Loc2 and Loc3, the algorithm found the correct allele assignments, but did not return them since some genotypes were inconsistent with those assignments due to meiotic error.

The results of `catalanAlleles` can be passed to `recodeAllopoly` in the same way as the results of `testAlGroups`.

5 Testing assignment accuracy using simulated datasets

The simulated data in this tutorial, as well as the simulations for the manuscript, were created using the `simAllopoly` function. If you have a different ploidy, number of individuals, or number of alleles from the datasets simulated in the manuscript, you can run your own simulations to estimate the accuracy of allele assignment. By default, the alleles are given names that start with A, B, *etc.* to indicate to which isolocus they belong, so that it is easy to see whether the output of `testAlGroups` is correct. See `?simAllopoly` for more information. The `tables_figs.R` file that is included as supplementary information for the manuscript can serve as a guide for how to run a large number of simulations and test their accuracy.