Assigning alleles to isoloci in POLYSAT

Lindsay V. Clark
University of Illinois, Urbana-Champaign

May 12, 2015

1 Introduction

This tutorial accompanies the R package POLYSAT versions 1.4 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, 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 1.4 Tutorial Manual". Additionally, there is a manuscript on BioRxiv at `http://dx.doi.org/10.1101/01xxxx` that describes in detail the rationale and limitations of the allele assignment tools described in this manual. 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
```

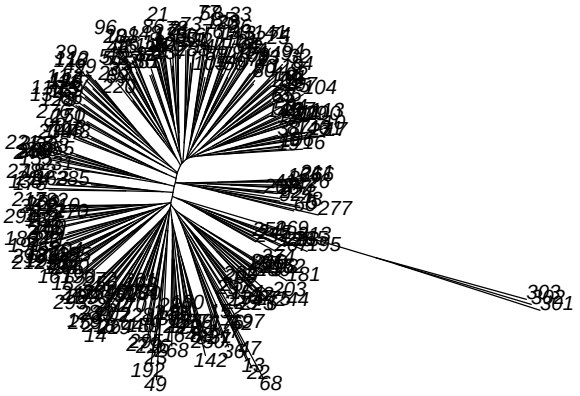
```
> Genotype(mydata, 43, 2) <- Missing(mydata)
> Genotype(mydata, 43, 2)
```

```
[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")
```

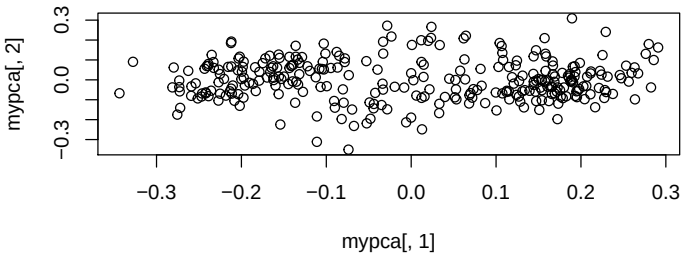


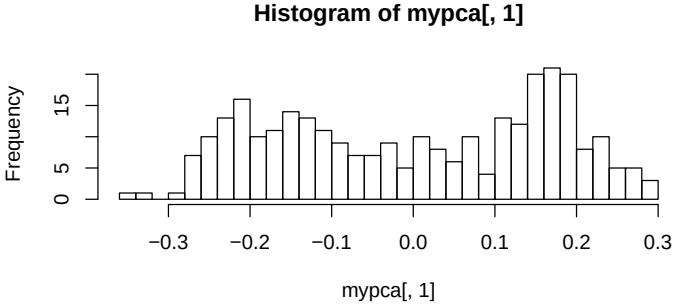
You can see that individuals 301, 302, and 303 are highly dissimilar from the rest. This is what it looks like when some individuals are a different species. Because of the fast rate at which microsatellites mutate, allele assignments that we make in one species are very unlikely to apply to another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

3 The polysat algorithm for allele assignment

3.1 General considerations and parameters

There are two functions in POLYSAT that are intended to be used one after the other for assigning alleles to isoloci. `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`
Homoplasy	`rare.al.check = 0`

Because you may not know whether the last three issues are present in your dataset, I recomment trying several parameter sets.

3.2 Running the algorithm

Below, I create a loop to run the algorithm on both populations and all seven loci, using a few different parameter sets. If your data are not tetraploid, please see `?alleleCorrelations` and `?testAlGroups` for how you should adjust the `n.subgen` and `SGploidy` arguments.

```
> nloc <- length(Loci(mydata)) # 7 loci
> # lists to contain results of alleleCorrelations
> CorrPop1 <- CorrPop2 <- list()
> length(CorrPop1) <- length(CorrPop2) <- nloc
> names(CorrPop1) <- names(CorrPop2) <- Loci(mydata)
> # lists to contain results of testAlGroups
> TAGpop1param1 <- list()
> length(TAGpop1param1) <- nloc
> names(TAGpop1param1) <- Loci(mydata)
> TAGpop1param2 <- TAGpop1param3 <- TAGpop1param1
> TAGpop2param1 <- TAGpop2param2  <- TAGpop2param3 <- TAGpop1param1
> # loop through loci
> for(L in Loci(mydata)){
+   # allele correlations
```

```
+   CorrPop1[[L]] <- alleleCorrelations(mydata, samples=pop1ind, locus = L)
+   CorrPop2[[L]] <- alleleCorrelations(mydata, samples=pop2ind, locus = L)
+   # default parameter set
+   TAGpop1param1[[L]] <- testAlGroups(mydata, CorrPop1[[L]], samples=pop1ind)
+   TAGpop2param1[[L]] <- testAlGroups(mydata, CorrPop2[[L]], samples=pop2ind)
+   # optimized for homoplasy
+   TAGpop1param2[[L]] <- testAlGroups(mydata, CorrPop1[[L]], samples=pop1ind,
+                                      rare.al.check=0)
+   TAGpop2param2[[L]] <- testAlGroups(mydata, CorrPop2[[L]], samples=pop2ind,
+                                      rare.al.check=0)
+   # optimized for null alleles
+   TAGpop1param3[[L]] <- testAlGroups(mydata, CorrPop1[[L]], samples=pop1ind,
+                                      null.weight=0)
+   TAGpop2param3[[L]] <- testAlGroups(mydata, CorrPop2[[L]], samples=pop2ind,
+                                      null.weight=0)
+ }

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.

```
> CorrPop1[["Loc6"]]$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

> CorrPop2[["Loc6"]]$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
```

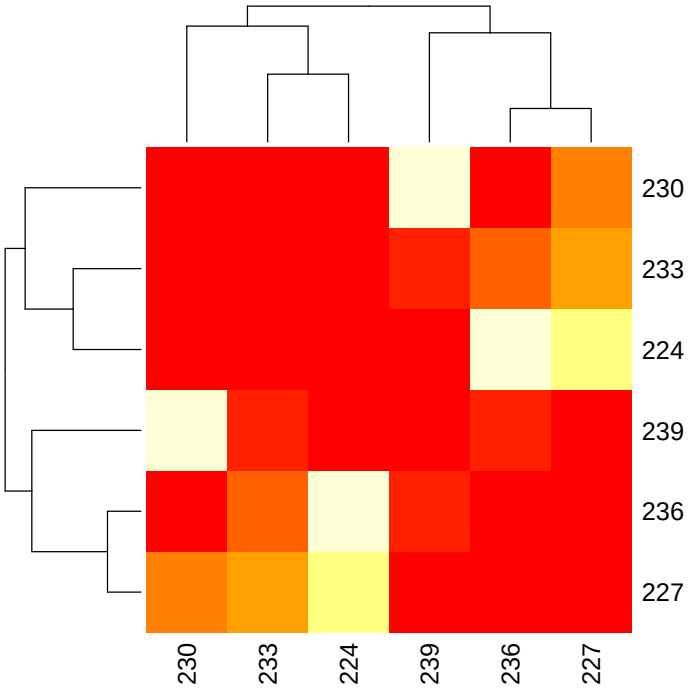
We see positive correlations between alleles 330 and 327, 339 and 336, and 348 and 345. Since these are trinucleotide repeats, it looks like some of the larger alleles (which would tend to have more stutter) had stutter peaks miscalled as alleles. If this were a real dataset, I would say to go back to 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")
```

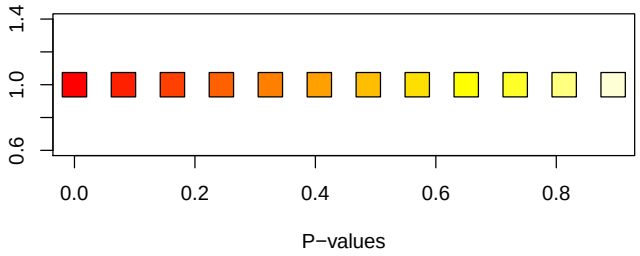
3.3.2 Quality of clustering

It is a good idea to manually inspect the results at each locus. Let's start with the first one.

```
> # Population 1
> heatmap(CorrPop1[["Loc1"]]$heatmap.dist, symm=TRUE)
```

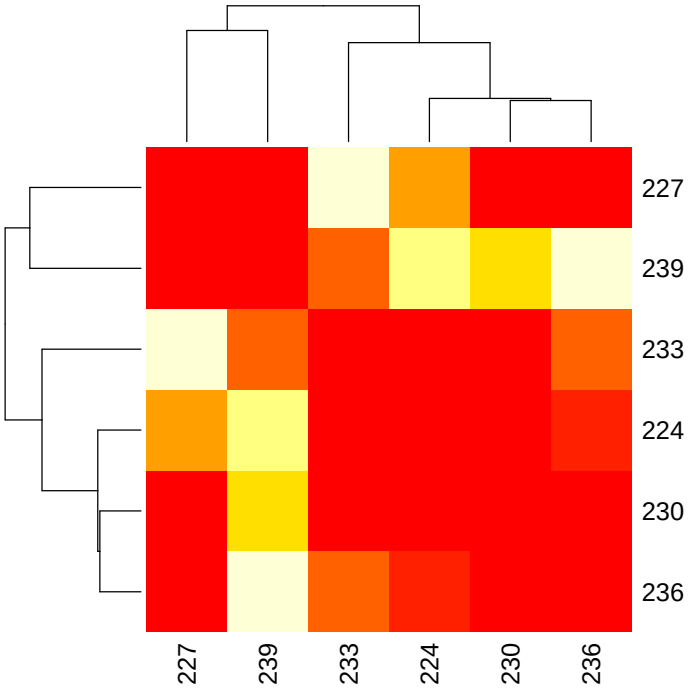


```
> # A plot to show how the colors correspond to p-values in the
> # heat map; you can repeat this for the other heat maps in this
> # tutorial if you wish.
> plot(x=seq(min(CorrPop1[["Loc1"]]$heatmap.dist),
+            max(CorrPop1[["Loc1"]]$heatmap.dist), length.out=12),
+      y=rep(1,12), xlab="P-values", ylab="", bg=heat.colors(12),
+      pch=22, cex=3)
```




```
> CorrPop1[["Loc1"]]$Kmeans.groups

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

> CorrPop1[["Loc1"]]$UPGMA.groups

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

> TAGpop1param1[["Loc1"]]$assignments

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

> TAGpop1param2[["Loc1"]]$assignments

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

> TAGpop1param3[["Loc1"]]$assignments

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

```
> # Population 2
> heatmap(CorrPop2[["Loc1"]]$heatmap.dist, symm=TRUE)
```




```
> CorrPop2[["Loc1"]]$Kmeans.groups

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

> CorrPop2[["Loc1"]]$UPGMA.groups

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

> TAGpop2param1[["Loc1"]]$assignments

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

> TAGpop2param2[["Loc1"]]$assignments
```

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

```
> TAGpop2param3[["Loc1"]]$assignments
```

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

Although the different assignments are not in complete agreement with each other among populations and parameter sets, we always see one homoplasious allele in the results from `testAlGroups`, which suggests that the second parameter set will be the most accurate. (Note that homoplasious alleles will never show up in the output of `alleleCorrelations`.) Using the second parameter set, identical assignments were produced in both populations, and those assignments are consistent with the clustering that we see in the heatmaps for both populations. Let's start a list of assignments that we will use when we recode the dataset.

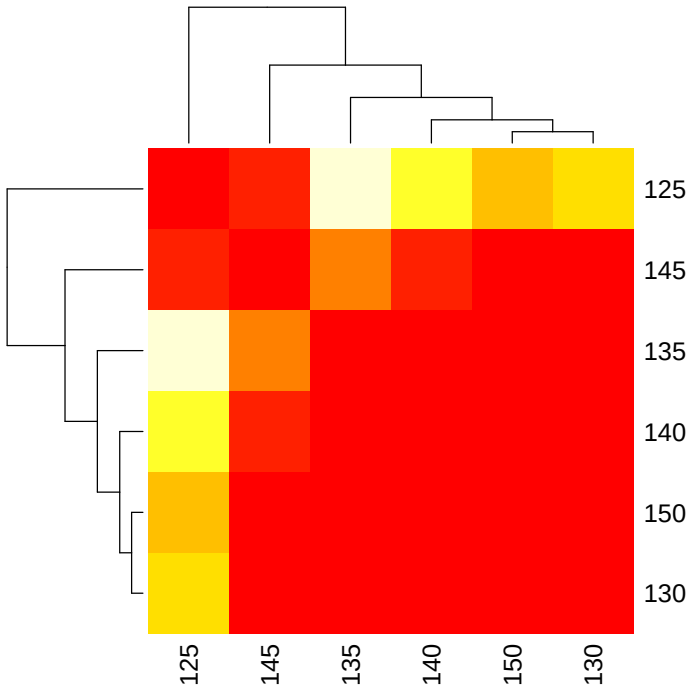
```
> AssignToUse <- list()
> AssignToUse[[1]] <- TAGpop1param2[["Loc1"]]
```

I will leave it as an exercise for the reader to inspect all of the remaining loci in the same way. Below are the assignment sets that I chose.

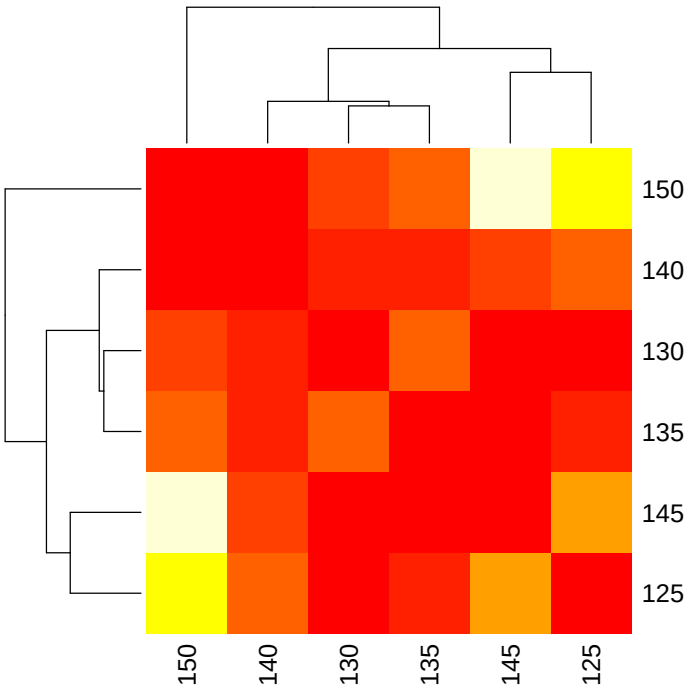
```
> AssignToUse[[2]] <- TAGpop1param1[["Loc2"]]
> AssignToUse[[3]] <- TAGpop1param1[["Loc3"]]
> AssignToUse[[4]] <- TAGpop1param1[["Loc4"]]
> AssignToUse[[5]] <- TAGpop1param1[["Loc5"]]
```

Loc7 looks a bit different from the others.

```
> heatmap(CorrPop1[["Loc7"]]$heatmap.dist, symm=TRUE)
```




```
> heatmap(CorrPop2[["Loc7"]]$heatmap.dist, symm=TRUE)
```




```
> TAGpop1param1[["Loc7"]]$assignments

     125 130 135 140 145 150
[1,]   1   1   1   0   0   1
[2,]   0   1   1   1   1   1

> TAGpop1param2[["Loc7"]]$assignments

     125 130 135 140 145 150
[1,]   1   1   1   0   0   1
[2,]   0   1   1   1   1   1

> TAGpop1param3[["Loc7"]]$assignments

     125 130 135 140 145 150
[1,]   1   1   1   0   0   1
[2,]   0   1   1   1   1   1

> TAGpop2param1[["Loc7"]]$assignments

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

```
> TAGpop2param2[["Loc7"]]$assignments

     125 130 135 140 145 150
[1,]   1   1   1   0   1   0
[2,]   0   1   0   1   1   1

> TAGpop2param3[["Loc7"]]$assignments

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

On the heatmaps, we don't see clustering of alleles into two distinct groups. When we look at allele assignments we see a lot of homoplasy, as well as disagreement between the results for the two populations. This is what it looks like when a marker is not actually segregating as multiple isoloci. When I simulated this locus, I 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). Since for our dataset it was just one locus, we will exclude this locus from analysis.

```
> mydata <- deleteLoci(mydata, loci="Loc7")
```

3.4 Merging assignments across populations

Although it was not necessary with this dataset, note that it is possible to merge allele assignments across data from two populations. This may be especially useful if some populations have alleles that are absent in other populations. For example, if we wanted to merge the different allele assignments from Loc1 using the first parameter set:

```
> Loc1Param1Merged <- mergeAlleleAssignments(list(TAGpop1param1[["Loc1"]],
+                                                 TAGpop2param1[["Loc1"]]))
> Loc1Param1Merged[[1]]$assignments

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

You can put assignments for multiple loci within the list that is passed to `mergeAlleleAssignments`, and the list that it returns will have one set of assignments per locus.

3.5 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.

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, AssignToUse)
> summary(recodedData)
```

```
Dataset with allele copy number ambiguity.
Simulated allotetraploid dataset.
Number of missing genotypes: 520
300 samples, 10 loci.
1 populations.
Ploidies: 2 3 1
Length(s) of microsatellite repeats: 3 4 5
```

You'll notice that we have twice as many loci now that each marker 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
```

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
```

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.

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
```

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`.